Studies on Disjunctive Logic Programming(Dissertation_全文)

AUTHOR(S):

Sakama, Chiaki

STUDIES ON DISJUNCTIVE LOGIC PROGRAMMING

by

CHIAKI SAKAMA

A dissertation submitted to
the Faculty of Engineering of
Kyoto University
for the degree of Doctor of Engineering

July 1994

Preface

Recent studies have enriched the expressive power of logic programming as a knowledge representation tool and the growing importance of logic programming in artificial intelligence is well recognized these days. Disjunctive logic programming is one of such extensions of logic programming which provides us with the ability of reasoning with indefinite information. Due to their expressiveness, disjunctive logic programming has been given an increasing attention over the past few years.

In this dissertation, we study theoretical frameworks for disjunctive logic programming. Our particular interest is in the semantic issues of disjunctive logic programs and their correspondences to commonsense reasoning in artificial intelligence.

As a semantics of disjunctive logic programs, we propose a new declarative semantics called the possible model semantics. The possible model semantics is an alternative theoretical framework for disjunctive logic programs, which provides a flexible inference mechanism for representing knowledge and also has a computational advantage over the classical minimal model semantics.

To relate disjunctive logic programs to commonsense reasoning in artificial intelligence, we propose transformations from disjunctive logic programs to various forms of nonmonotonic reasoning such as default logic, circumscription, and autoepistemic logic. Moreover, we discuss connections between disjunctive logic programs and abductive logic programs, and reveal close relationships between each framework.

Another important issue for commonsense reasoning is the treatment of inconsistent knowledge. Since classical logic programming is not useful in inconsistent programs, we introduce paraconsistent semantics for disjunctive logic programs which provide uniform frameworks for handling both indefinite and inconsistent information in a program.

We finally discuss program optimization issues in disjunctive logic programs. A technique of partial deduction in logic programming is extended to disjunctive logic programs and its correctness is presented.

Acknowledgments

I would like to express my sincere gratitude to my thesis advisor, Shuji Doshita, for his continuous guidance and encouragement. I would also like to thank the judging committee members, Yoshihiro Matsumoto and Toru Ishida.

This research was started when I was at ICOT (Institute for New Generation Computer Technology), and was continued at ASTEM (Advanced Software Technology and Mechatronics Research Institute of Kyoto). I would like to thank Kazuhiro Fuchi and Yutaka Ohno for providing me with opportunities to pursue this research. My appreciation also goes to my research managers, Hidenori Itoh, Shun-ichi Uchida, and Koichi Furukawa for their helpful advice and encouragement.

Special thanks are due to Susumi Yamasaki, who guided me to the area of logic programming when I was at Kyoto University. I am also grateful to my research colleagues for their fruitful discussions and valuable comments. Though I cannot list all their names, my special thanks go to Katsumi Inoue and Hirohisa Seki.

Discussions with researchers in the areas of logic programming and nonmonotonic reasoning are always stimulating and helpful. In particular, I would like to thank Edward P. F. Chan, Hendrik Decker, Jürgen Dix, Michael Gelfond, Georg Gottlob, John Grant, Jack Minker, and V. S. Subrahmanian for useful correspondence on the work presented in this dissertation.

Contents

Chapter 1

Introduction

All parties to the debate agree that a central goal of research is that computers must somehow come to "know" a good deal of what every human being knows about the world and about organisms, natural and artificial, that inhabit it. This body of knowledge – indefinite, no doubt, in its boundaries – goes by the name "common sense".

— David Israel [1983].

After the discovery of *resolution principle* by J. Alan Robinson [1965a], logic based approaches to *artificial intelligence* (AI) have been developed in the area of automated theorem proving in the late 1960's [Chang and Lee, 1973]. In the early 1970's, Kowalski [1974] attached a procedural interpretation to *Horn clause logic* and introduced the framework of *logic programming*. This leads to the design of the programming language PROLOG by Colmerauer and his group [1973]. After the success of PROLOG, the logic programming framework has been employed in many AI projects in the world, including the Japanese *Fifth Generation Computer Project* in the 1980's.

A prominent feature of logic programming is its formal logical basis. That is, a logic program is viewed as a set of axioms, its computation corresponds to deduction, and the meaning of a program is exactly the logical consequences of the program. It is worth noting that such a formal foundation of logic programming enables researchers in this field to communicate with each other using mathematical logic as a common language. This is an important and distinguished advantage of logic programming which is never seen in the fields of expert systems nor object-oriented databases today. This unique feature of logic programming has promoted theoretical studies in its own right, and found various applications ranging from databases to artificial intelligence [Gallaire et al., 1984; Kowalski, 1991; Baral and Gelfond, 1994].

Another important feature of logic programming is the separation of logic and control [Kowalski, 1979]. That is, logic programs specify declarative sentences representing knowledge in the world, while their algorithms are designed independent of the contents of programs. Such a separation is also important from the viewpoint of knowledge representation in artificial intelligence [McCarthy and Hayes, 1969].

A declarative meaning of a logic program is characterized by the *declarative semantics*, which is usually given by the model theory of first-order logic. On the other hand, a computational aspect of a logic program is characterized by the *procedural semantics*, which is provided as a proof procedure of the program. Thus, declarative and procedural semantics characterize two different aspects of logic programming.

The semantics of logic programming was firstly studied by van Emden and Kowalski [1976], in which they introduced declarative and procedural semantics of *Horn logic programs*. Horn logic programs are the simplest class of logic programming and have applications such as the programming language PROLOG or *deductive databases* [Gallaire et al., 1984].

Horn logic programs have powerful computational mechanisms in the sense that they are as expressive as a Turing machine [Tärnlund, 1977]. Considering Horn logic programs as a knowledge representation language, however, they describe only definite information in the world and provide no inference mechanism for reasoning with *incomplete* information. In fact, these limitations have often caused a criticism that knowledge representations based on formal logics and deductive inference are not useful for commonsense reasoning in AI.

The necessity of dealing with incomplete information in knowledge representation is addressed by Levesque [1983]:

> The reason incomplete knowledge hases are so important is that, in many applications, the knowledge base undergoes a continual evolution. At each stage, information can be acquired that is potentially very vague or indefinite in nature. More important, a problem solving system cannot simply wait for the knowledge base to stabilize in some final and complete form since this may never happen.

Then, in order to overcome those limitations, attempts have been done to enhance the expressiveness and inference abilities of logic programming. Such extensions are mainly achieved in two ways.

First extension is the representation of *negation*. In a definite Horn logic program, each fact derived from the program represents true knowledge in the program, while any fact unproved in the program is assumed to be false as *default negation* or *negation*

as failure to prove [Clark, 1978; Reiter, 1978]. Such default negation can augment incomplete knowledge in a program, while it makes logic programming semantics *nonmonotonic*. Together with this non-classical mechanism, the framework of logic programming was extended from Horn logic programs to non-Horn logic programs by incorporating default negation in programs. *Stratified logic programs* [Chandra and Harel, 1985; Apt et al., 1988; van Gelder, 1988] and *normal logic programs* [Gelfond and Lifschitz, 1988; van Gelder et al., 1991] are such extensions. Those frameworks were further extended by Gelfond and Lifschitz [1990] to *extended logic programs* which include *classical negation* as well as default negation.

Second extension is the representation of *indefinite* or *disjunctive information*. Logic programming which can represent such information is called *disjunctive logic programming*. A theoretical framework of disjunctive logic programming was firstly studied by Minker [1982] for *positive disjunctive programs*, and then generalized to *normal disjunctive programs* [Lobo et al., 1992], and *extended disjunctive programs* [Gelfond and Lifschitz, 1991] by including negation.

Negation and disjunctions are two important extensions of logic programming which provide abilities to deal with incomplete information in a program. Those extensions of logic programming greatly increase the expressive power of logic programming as a knowledge representation tool and realize commonsense reasoning in artificial intelligence.

With these backgrounds, in this dissertation we study theoretical frameworks of disjunctive logic programming.

1.1 Motivations and Objectives

Most of the semantics of logic programming have traditionally been studied based on the notion of *minimal models*. The notion of minimal models reflects the so-called *Occam's razor*[1] in the sense that a minimal model contains exactly as many facts as required to hold in a program. Such a *principle of minimality* has been supported in the area of not only logic programming, but also *nonmonotonic logics* in artificial intelligence. Therefore, it has been recognized that the principle of minimality is one of the most basic and indispensable criteria that each semantics for commonsense reasoning should obey [Schlipf, 1992a]. In fact, the *least Herbrand model semantics* of Horn logic programs [van Emden and Kowalski, 1976], the *perfect model semantics* of stratified logic programs [Przymusinski, 1988a], and the *stable model semantics* of normal

[1] William of Occam, 1285-1349?, England, Scholastic Philosopher.

logic programs [Gelfond and Lifschitz, 1988] are all based on the principle of minimality. This is also the case in the context of disjunctive logic programming, namely, the *minimal model semantics* of positive disjunctive programs [Minker, 1982] and the *disjunctive stable model semantics* of normal disjunctive programs [Przymusinski, 1991a; Gelfond and Lifschitz, 1991] are both minimal.

However, such a minimalism is not always appropriate in a program containing indefinite information. Ross and Topor [1988] have firstly noticed this problem in the context of inferring negation in disjunctive logic programs. They argue that when one infers negation from a disjunctive logic program, one should be cautious to interpret disjunctions *inclusively* rather than *exclusively*. In fact, the minimal model semantics minimizes truth facts, then it usually interprets disjunctions exclusively and maximizes negative information inferred from a program.

Then Ross and Topor gave a framework for inferring negation in inclusive disjunctive logic programs, which is different from the minimal model semantics. However, the problem is that their framework, on the contrary, has difficulty for treating exclusive disjunctions in a program. Moreover, they provided no model theoretical meaning for inclusive disjunctive logic programs as a counterpart of the minimal model semantics. Thus our first objective is to give a theoretical framework of disjunctive logic programs which can distinguish both exclusive and inclusive disjunctions uniformly in a program.

Our second objective is to clarify the relations between disjunctive logic programs and various forms of commonsense reasoning in artificial intelligence. Recent studies have revealed close relationships between logic programming and *nonmonotonic reasoning* [Reiter, 1982; Lifschitz, 1985; Gelfond, 1987; Lifschitz, 1988; Przymusinski, 1988b; Gelfond et al., 1989; Bidoit and Froidevaux, 1991a; Bidoit and Froidevaux, 1991b], and there are increasing interests to investigate interrelations between the two areas [Nerode et al., 1991; Pereira and Nerode, 1993]. These studies are important both for logic programming and artificial intelligence. Representing logic programs in terms of nonmonotonic formalisms helps us to realize commonsense reasoning in logic programming, on the other hand, it opens possibilities for using logic programming proof procedures as inference engines for nonmonotonic reasoning. Thus, clarifying such interrelations enables us to use techniques developed in one area by the other, and it will help researchers in each community to progress their work and enrich perspectives. Until now, those interrelations have been mainly studied for normal logic programs, then our next goal is to extend those previously studied results to the case of disjunctive logic programs.

Abduction is also a form of commonsense reasoning to which much attention has been paid recently. Abduction supplies an ability to perform reasoning with hypotheses, and its growing importance in various AI problems is widely recognized. In the context of logic programming, abduction is realized by the framework of *abductive logic programming* [Kakas et al., 1992]. Abductive logic programming is an extension of logic programming and also realizes reasoning with incomplete information in a program. In this regard, abductive logic programs and disjunctive logic programs appear to deal with very similar problems from different viewpoints. However, each formalism has been independently developed so far and has different syntax and semantics from the other. Then our primary interest is whether there is any formal correspondences between those two frameworks.

Another important aspect of commonsense reasoning in logic programming is the treatment of inconsistent knowledge. In practical situations, inconsistency is likely to happen as well as indefiniteness when we build a large scale of knowledge base. In such a knowledge base, it may be the case that a program contains local inconsistency and yet might have a global intended meaning. However, as traditional logic programming is based on classical first-order logic, a piece of inconsistent information makes a whole program trivial. A logic which is not destructive in the presence of inconsistent information is called *paraconsistent logic*, and its application to logic programming is known as *paraconsistent logic programming* [Blair and Subrahmanian, 1989]. Then, in order to treat both inconsistent and indefinite information in a program, paraconsistent frameworks for disjunctive logic programs are necessary.

Our last objective is an optimization issue of disjunctive logic programs. In logic programming, program transformation and optimization are important from practical viewpoints since correctly specified programs are not necessarily efficient programs. Partial deduction or partial evaluation is known as one of such optimization techniques in logic programming. Partial deduction derives a more specific program through performing deduction on a part of the program, while retaining the original meaning of the program. Such a specialized program is usually more efficient than the original program when executed. Partial deduction techniques have been mainly studied for normal logic programs so far. However, since computation of disjunctive logic programs is generally expensive than normal logic programs, it is necessary to develop partial deduction techniques for disjunctive logic programs also.

To summarize, the objectives of this dissertation are to study various aspects of disjunctive logic programming from both theoretical and practical points of view.

1.2 Contributions

The contributions of the research reported in this dissertation are as follows.

1. We propose a new declarative semantics of disjunctive logic programs called the *possible model semantics*. The possible model semantics is a different theoretical framework from the classical minimal model semantics, and provides a flexible negative inference mechanism under the closed world assumption. A new fixpoint semantics of disjunctive logic programs is introduced to characterize the possible model semantics and its proof procedure is presented. It is shown that the possible model semantics also has a computational advantage over the minimal and the stable model semantics of disjunctive logic programs.

2. We show a method of representing disjunctive logic programs in terms of *default logic* [Reiter, 1980]. The problem of previously proposed approaches is pointed out, and an alternative correct transformation from disjunctive logic programs to default theories is proposed. We also present the correspondences between disjunctive logic programs and other forms of nonmonotonic reasoning in AI such as *disjunctive default logic* [Gelfond et al., 1991], *circumscription* [McCarthy, 1980], and *autoepistemic logic* [Moore, 1985].

3. We reveal a close relationship between disjunctive logic programs and abductive logic programs. It is shown that the possible models of disjunctive logic programs are essentially equivalent to the *generalized stable models* of abductive logic programs [Kakas and Mancarella, 1990]. We also show that the possible model semantics is useful to characterize abductive logic programs from the computational complexity point of view.

4. We propose paraconsistent frameworks for disjunctive logic programs which can distinguish inconsistent information from other information in a program. Paraconsistent semantics for extended disjunctive programs are introduced which are based on lattice-structured multi-valued logics. Fixpoint characterizations of those paraconsistent semantics are presented, and methods for reasoning with inconsistency are discussed.

5. We develop partial deduction techniques for disjunctive logic programs. In disjunctive logic programs, normal partial deduction is shown to be not useful, and a new partial deduction method for disjunctive logic programs is introduced. It is shown that the proposed partial deduction preserves the meanings of disjunctive logic programs, and its application to goal-oriented partial deduction is presented for query optimization.

1.3 Outline of the Dissertation

This dissertation consists of eight chapters. The rest of the dissertation is organized as follows.

In Chapter 2, we first give basic notions of logic programming and other related notions used in this dissertation. We introduce the framework of disjunctive logic programming and review previously studied results on logic programming and disjunctive logic programming.

In Chapter 3, we propose a new theoretical framework of disjunctive logic programs. The *possible model semantics* is introduced as a declarative semantics of disjunctive logic programs. Negative inference under the possible model semantics is presented and its properties are discussed. A new fixpoint semantics of disjunctive logic programs is introduced to characterize the possible model semantics. For the procedural part, a bottom-up proof procedure is provided to compute the possible model semantics of positive and normal disjunctive programs. We also compare computational complexities of various semantics of disjunctive logic programs and show that the possible model semantics has a computational advantage over other proposed semantics.

In Chapter 4, we present a method of relating disjunctive logic programs to Reiter's default logic. We first point out the problem of previously studied results, then establish a correct default translation of disjunctive logic programs. We show a one-to-one correspondence between the stable models of a disjunctive logic program and the extensions of its associated default theory. Next we extend the results to extended disjunctive programs and investigate the connection with Gelfond et al.'s disjunctive default logic. The stable model semantics of disjunctive logic programs is also characterized by Moore's autoepistemic logic and McCarthy's circumscription. We also show that the possible model semantics is naturally expressed in terms of autoepistemic logic.

In Chapter 5, we present the equivalence relationship between disjunctive logic programs and abductive logic programs. We show that the generalized stable models of abductive logic programs can be translated into the possible models of disjunctive logic programs, and vice versa. It is also shown that abductive disjunctive programs can be expressed by abductive logic programs under the possible model semantics. Moreover, we observe that when considering the disjunctive stable model semantics instead of the possible model semantics, it is unlikely that disjunctive logic programs can be efficiently expressed in terms of abductive logic programs. The usefulness

of the possible model semantics for abductive logic programs is also verified by its computational complexity.

In Chapter 6, we propose declarative semantics of possibly inconsistent disjunctive logic programs. We introduce the *paraconsistent minimal and stable model semantics* for extended disjunctive programs, which can distinguish inconsistent information from other information in a program. The possible model semantics introduced in Chapter 3 is also extended to the *paraconsistent possible madel semantics* in extended disjunctive programs. These semantics are natural extensions of the *answer set semantics* of extended disjunctive programs [Gelfond and Lifschitz, 1991], and are based on lattice-structured multi-valued logics. Paraconsistent semantics are also characterized by a fixpoint semantics of extended disjunctive programs, and methods for reasoning in inconsistent programs are addressed.

In Chapter 7, we present methods of partial deduction for disjunctive logic programs. We first show that normal partial deduction is not useful for disjunctive logic programs, then introduce *disjunctive partial deduction* for disjunctive logic programs. It is proved that disjunctive partial deduction preserves the minimal model semantics of positive disjunctive programs and the disjunctive stable model semantics of normal disjunctive programs. We also show that together with suitable program transformations, normal partial deduction can compute disjunctive partial deduction and also preserves the possible model semantics. The proposed partial deduction method is applied to goal-oriented partial deduction for query optimization.

In Chapter 8, we conclude the dissertation and address future directions of the research.

1.4 Publications

Some chapters of this dissertation are based on the published papers as follows.

Chapter 3 is based on the papers [Sakama, 1989] and [Sakama and Inoue, 1993a], which were presented at the *First International Conference on Deductive and Object-Oriented Databases* (DOOD'89; Kyoto, Japan, December 1989) and the *Tenth International Conference on Logic Programming* (ICLP'93; Budapest, Hungary, June 1993), respectively. Also much of this chapter is in [Sakama and Inoue, 1994c] which will appear in the *Journal of Automated Reasoning*.

Chapter 4 is based on the paper [Sakama and Inoue, 1993b] which was presented at the *Second International Workshop on Logic Programming and Nonmonotonic Reasoning* (LPNMR'93; Lisbon, Portugal, June 1993).

Chapter 5 is based on the paper [Sakama and Inoue, 1994a] which was presented at the *Eleventh International Conference on Logic Programming* (ICLP'94; S. Margherita Ligure, Italy, June 1994).

Chapter 6 is based on the paper [Sakama and Inoue, 1994b] which will appear in the *Journal of Logic and Computation*. Related topic is also discussed in the paper [Sakama, 1992] which was presented at the *International Conference on Fifth Generation Computer Systems* (FGCS'92; Tokyo, Japan, June 1992).

Chapter 7 is based on the paper [Sakama and Seki, 1994] which was presented at the *Fourth International Workshop on Logic Program Synthesis and Transformation* (LOPSTR'94; Pisa, Italy, June 1994). Related topic is also presented in the paper [Sakama and Itoh, 1988] which was published in the *Journal of New Generation Computing*, vol. 6, Nos.2-3.

The dissertation does not include those topics presented in author's published papers [Sakama and Itoh, 1988] and [Sakama and Okumura, 1988]. These papers also discuss theoretical aspects of logic programming and nonmonotonic reasoning from different viewpoints.

Chapter 2

Preliminaries

This chapter introduces basic notions and terminologies used in this dissertation and reviews previously studied results on logic programming.

2.1 First-Order Logic

A framework of logic programming stems from classical first-order logic. Then we start from a brief overview of first-order theories. Terminologies and notations presented in this section are based on [Lloyd, 1987; Apt, 1990].

2.1.1 Language

An *alphabet* of a *first-order language* consists of a set of *constants*, *variables*, *function symbols*, *predicate symbols*, and usual punctuation symbols, together with connectives $\wedge$, $\vee$, $\neg$, $\supset$,[1] $\equiv$ and quantifiers $\exists$ and $\forall$.

A *term* is defined inductively as either a variable or a constant or an expression of the form $f(t_1, \ldots, t_n)$ where f is a function symbol and t_i's are terms.

A *formula* is defined inductively as follows:

(i) An *atom* $p(t_1, \ldots, t_n)$ is a formula where p is a predicate symbol and t_i's are terms.

(ii) For formulas F and G, $\neg F$, $F \vee G$, $F \wedge G$, $F \supset G$, and $F \equiv G$ are all formulas.

(iii) For a formula F and a variable x, $\exists x\, F$ and $\forall x\, F$ are formulas.

[1] In the context of logic programming, the connective $\leftarrow$ is also used for implication.

A formula is *closed* if it contains no free occurrences of any variable (that is, any variable is bounded by some quantifier). A term or a formula containing no variables is called *ground*. A ground formula is also called a *proposition* or a *propositional formula*. A first order language L over an alphabet is defined as the set of all formulas constructed from the symbols of the alphabet. A *literal* is an atom A or its negation $\neg A$. A literal is called *positive* if it is an atom, otherwise it is called *negative*.

A *clause* is a formula of the form:

$$\forall x_1, \ldots, \forall x_k \, (A_1 \vee \ldots \vee A_l \vee \neg B_1 \vee \ldots \vee \neg B_m)$$

where each A_i $(1 \leq i \leq l)$ and B_j $(1 \leq j \leq m)$ are atoms and $x_1, \ldots, x_k$ are variables occurring in the formula. A clause is also written as:

$$\forall x_1, \ldots, \forall x_k \, (A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m)$$

or simply as:

$$A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m. \tag{2.1}$$

The left-hand side of the clause (2.1) is called the *head*, and the right-hand side of the clause is called the *body*. Note that any closed first-order formula can be transformed to clausal form [Chang and Lee, 1973].

2.1.2 Model Theory

The meaning of a first-order theory is given by an interpretation of formulas.

An *interpretation* I for a first-order language L consists of a non-empty set D, called the *domain* of I, and an *assignment* defined as follows:

- each constant in L is assigned to an element in D,
- each n-ary function in L is assigned to a mapping from D^n to D,
- each n-ary relation r in L is assigned to a mapping from D^n to $\{true, false\}$,

where $true$ and $false$ are the *truth values* assigned to propositions.

Given an interpretation I of L, a *variable assignment* V is an assignment to each variable in L of an element in the domain of I. The *term assignment* (with respect to I and V) of the terms in L is defined as follows:

- for each variable in L, the assignment according to V,
- for each constant in L, the assignment according to I,

- for each n-ary function $f(t_1, \ldots, t_n)$ in L, the assignment $f'(t'_1, \ldots, t'_n)$ where f' is the assignment of f according to I, and $t'_1, \ldots, t'_n$ are the term assignments of $t_1, \ldots, t_n$.

Now given an interpretation I and a variable assignment V, truth values of formulas are inductively defined as follows.

(i) If the formula is an atom $p(t_1, \ldots, t_n)$, then its truth value is obtained by calculating the value of $p'(t'_1, \ldots, t'_n)$, where p' is the mapping assigned to p by I and $t'_1, \ldots, t'_n$ are the term assignments of $t_1, \ldots, t_n$ with respect to I and V.

(ii) Given formulas F and G,

- $\neg F$ is true (resp. false) iff F is false (resp. true).
- $F \vee G$ is true (resp. false) iff either F or G is true (resp. both F and G are false).
- $F \wedge G$ is true (resp. false) iff both F and G are true (resp. either F or G is false).
- $F \supset G$ is true (resp. false) iff either F is false or G is true (resp. both F is true and G is false).
- $F \equiv G$ is true (resp. false) iff both $F \supset G$ and $G \supset F$ are true (resp. either $F \supset G$ or $G \supset F$ is false).

(iii) The formula $\exists x \, F$ is true iff there exists $d \in D$ such that F has a truth value true with respect to I and $V(x/d)$, where $V(x/d)$ is V except that x is assigned to d; otherwise its truth value is false.

(iv) The formula $\forall x \, F$ is true iff for all $d \in D$, F has a truth value true with respect to I and $V(x/d)$; otherwise its truth value is false.

A formula F is *satisfied* in an interpretation I (written $I \models F$) iff F is true in I for any variable assignment V. In particular, when F is closed, the truth value of the formula is independent of V. Given a set of formulas S, an interpretation I is called a *model* of S if I satisfies every formula in S. A set of formulas S is called *satisfiable* or *consistent* if it has a model; otherwise S is called *unsatisfiable* or *inconsistent*. When every interpretation is a model for S, we say that S is *valid*.

2.2 Logic Programming

In computer science, logic plays an important role for designing, analyzing, and reasoning about computer programs. On the other hand, *logic programming*, advocated by Kowalski [1974], uses first-order predicate logic itself as a procedural programming language. Logic programming has a very simple syntax and a clear semantics based on the formal mathematical logic and the theory of resolution proof procedures. Compared with conventional programming languages, logic programming provides a very high-level specification language for describing problems, and at the same time, it serves as a computer executable programming language.

In this section, we review the framework of logic programming and present previously studied resnlts (not necessarily in a comprehensive manner).

2.2.1 Horn Logic Programs

An early stage of logic programming is defined as a subset of first-order clauses of the form (2.1), called *Horn clauses* [Kowalski, 1974].

A *Horn logic program* is a finite set of Horn clauses of the form:

$$A_l \leftarrow B_1 \wedge \ldots \wedge B_m \ \ (0 \leq l \leq 1; m \geq 0) \tag{2.2}$$

where A_l and B_i's are atoms.

A Horn clause with a non-empty head ($l = 1$) is called a *definite clause*, and a definite clause with the empty body ($m = 0$) is called a *unit clause*. A Horn clause with the empty head ($l = 0$) and a non-empty body ($m \neq 0$) is called a *negative clause*, which is also called a *goal* or an *integrity constraint* depend on its usage. A Horn logic program including only definite clauses is called a *definite logic program*.

From the database point of view, a set of Horn clauses attaches deductive capabilities to conventional relational databases in the context of *deductive databases* [Gallaire et al., 1984]. For example, in a definite logic program it is easy to specify the transitive closure of a relation by *recursive* clauses like that:

$$ancestor(x, y) \leftarrow parent(x, z) \wedge ancestor(z, y),$$
$$ancestor(x, y) \leftarrow parent(x, y),$$

which cannot be expressed in relational databases [Ullman, 1982].

The declarative semantics of logic programming is usually defined by means of particular models of a program, called *Herbrand models*.

Given a logic program P, its *Herbrand universe* U_P is the set of all ground terms, which can be formed out of the constants and function symbols appearing in the language L of P.[2] The *Herbrand base* $\mathcal{HB}_P$ is the set of all ground atoms formed by predicate symbols from L with ground terms from U_P as arguments.

An *Herbrand interpretation* for P is the interpretation which is defined over the Herbrand universe U_P as a domain with an assignment as follows:

- each constant in L is assigned to itself in U_P,
- each n-ary function symbol f in L is assigned to the mapping from $(U_P)^n$ to U_P defined by $(t_1, \ldots, t_n) \rightarrow f(t_1, \ldots, t_n)$,
- each n-ary relation r in L is assigned to a mapping from $(U_P)^n$ to $\{true, false\}$.

Note that since the assignment to constants and function symbols is fixed in Herbrand interpretations, any Herbrand interpretation is identified with a corresponding subset of the Herbrand base.

An *Herbrand model* for P is an Herbrand interpretation which is a model for P. Throughout the dissertation, an interpretation means an Herbrand interpretation and a model means an Herbrand model. A consistent program has an Herbrand model, while an inconsistent program has none.

A *substitution* is a finite mapping from variables to terms $\sigma = \{x_1/t_1, \ldots, x_n/t_n\}$, where each x_i is a variable, each t_i is a term distinct from x_i, and the variables $x_1, \ldots, x_n$ are distinct. The substitution σ is called a *ground substitution* if all t_i's are ground. If σ is empty, it is called an *empty substitution*.

By an *expression* we mean a term, a literal, or a clause and denote it by E. Given an expression E and a substitution σ, the expression $E\sigma$ presents an *instance* of E by σ, which is obtained from E by simultaneously replacing each occurrence of the variable in E with the corresponding term according to σ. An instance is called *ground* if it contains no variable.

A substitution σ is called a *unifier* of expressions E and F iff $E\sigma = F\sigma$ holds. In particular, a unifier σ is called the *most general unifier (mgu)* iff for any unifier θ such that $E\theta = F\theta$, there is a substitution λ such that $\theta = \lambda\sigma$.

Given a logic program P and a clause C in P, an *Herbrand instantiation* of C is the set of all ground instances of C by any substitution σ which replaces every variable in C with an element from U_P. The *ground program* of P is the (possibly infinite) set of all Herbrand instantiations of every clause from P. A logic program

[2] We usually consider constants and function symbols appearing in a program unless stated otherwise. If the program contains no constants, some constant is added to form ground terms.

P has an Herbrand model M iff its ground program has the Herbrand model M. Thus, any logic program P is semantically identified with its ground program, so we usually assume ground programs in this dissertation when considering semantics of logic programs.

The declarative semantics of Horn logic programs is defined as the smallest Herbrand model, called the *least Herbrand model*. The least Herbrand model is also obtained as the intersection of all Herbrand models of the program [van Emden and Kowalski, 1976]. A definite logic program is always consistent and has the least Herbrand model.

The procedural semantics of logic programming is defined by proof procedures in two-ways. A *bottom-up* proof procedure is based on usual *modus ponens* and starts from the given facts and proceeds forward to the conclusions. On the other hand, Kowalski [1974] introduced a procedural interpretation of Horn logic programs in which the head of a definite clause is viewed as a procedure name and the body of the clause is viewed as a set of procedural calls. A *top-down* proof procedure of logic programming is based on this idea. It starts from a given goal to be proved and goes backward to premises in a program by iteratively producing its subgoals.

For definite logic programs, van Emden and Kowalski [1976] present bottom-up *hyperresolution* and show that the set of all ground atoms derived from a definite logic program by hyperresolution coincides with the least Herbrand model of the program. The least Herbrand model is also characterized by the *success set* of top-down *SLD-resolution* [Apt and van Emden, 1982]. Various proof procedures for (function-free) definite logic programs are also studied in the context of query-answering in deductive databases [Bancilhon and Ramakrishnan, 1988].

The least Herbrand model assigns the truth value *true* to each atom included in the model, and remained other atoms are interpreted as false. Proof-theoretically speaking, however, definite logic programs allow to derive only positive consequences and negative facts are never derived from a program. To infer negative information in a program, we can use negative clauses to represent false facts in a Horn logic program. However, it is impractical to represent negative information explicitly in a program, since a program usually specifies true sentences in the actual world while other false facts not written in the program are relatively huge (possibly infinite) in their amount. Then it is more convenient to consider a collection of definite clauses representing known positive information, and assume any fact non-derivable from the program to be false by *negation as failure to prove*. Such negation is also called *default negation* and is distinguished from classical negation in first-order logic.

To formalize an inference rule for default negation in logic programming, Reiter [1978] introduced the *closed world assumption* (CWA). Given a program P and for any ground atom A, the CWA is formally stated as follows:

$$CWA(P) \models \neg A \text{ iff } P \not\models A.$$

An alternative formalization of default negation is known as Clark's *program completion* [Clark, 1978]. Program completion characterizes the *finite failure set* of SLD-resolution, while it has some drawbacks compared with the CWA [Shepherdson, 1984; 1988].

Note that default negation in logic programming is *nonmonotonic* in its nature. That is, an addition of a new fact might withdraw a previously presumed default fact. It is known that such nonmonotonic reasoning is useful for commonsense reasoning in artificial intelligence. Thus introduction of default negation enhances inference abilities of logic programming beyond monotonic deduction, while it makes the theory of logic programming depart from classical first-order logic.

2.2.2 Normal Logic Programs

A (definite) Horn logic program is the simplest form of logic programming, however, as a knowledge representation language, its expressive power is very limited since it only allows one to specify conjunctions and implications between relations. With this restriction, we cannot compute even the complement of a given relation. We have already seen that default negation introduces a mechanism of negative inference into Horn logic programs. Then the first extension of logic programming is to incorporate such negation into a program and use it during deductive inference. To express default negation in a program, Horn logic programs are extended to include *normal clauses* as follows.

A *normal logic program* is a finite set of normal clauses of the form:

$$A_l \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\, B_{m+1} \wedge \ldots \wedge not\, B_n \quad (0 \leq l \leq 1;\ n \geq m \geq 0) \qquad (2.3)$$

where A_l and B_i's are atoms and *not* is the *negation-as-failure* operator. A normal clause with the empty head and a non-empty body is called an *integrity constraint*.

Note here that negation in normal clauses represents default negation and is distinguished using a new connective *not*. The above normal clause (2.3) is read as that A_l is true if $B_1, \ldots, B_m$ are true and $B_{m+1}, \ldots, B_n$ are not true. Clearly, normal logic programs reduce to Horn logic programs in the absence of default negation.

A normal logic program is called *stratified* [Chandra and Harel, 1985; Apt et al., 1988; van Gelder, 1988] if it contains no predicate derived recursively through its negation. More precisely, a normal logic program P is stratified if there is a partition $P = P_1 \cup \ldots \cup P_n$ such that the following conditions hold for $i = 1, \ldots, n$:

(i) If a predicate occurs positively in a clause in P_i, then its definition (i.e, clauses containing the predicate in their heads) is contained within $\bigcup_{j \leq i} P_j$.

(ii) If a predicate occurs negatively in a clause in P_i, then its definition is contained within $\bigcup_{j<i} P_j$.

P_1 can be empty.

For example, the popular flying-bird example is written in a stratified logic program like that:

$$fly(x) \leftarrow bird(x) \wedge not\ abnormal(x),$$
$$bird(x) \leftarrow penguin(x),$$
$$abnormal(x) \leftarrow penguin(x),$$

which means that a bird flies if she is not an abnormal bird.

The notion of stratified logic programs is further extended to *locally stratified logic programs* [Przymusinski, 1988a] which contain no self-recursive atom through negation, and *call-consistent/order-consistent logic programs* [Kunen, 1989; Sato, 1990] which contain no self-recursive predicate/atom through an odd number of negation.

In contrast to stratified logic programs, possibly unstratified normal logic programs are often called *general logic programs* [van Gelder, 1988; Gelfond and Lifschitz, 1988; van Gelder et al., 1991]. The following *game program* [van Gelder et al., 1991] is an example of such programs:

$$winning(x) \leftarrow move(x, y) \wedge not\ winning(y),$$

which presents a game situation that one is in a winning position if there is a move to a losing position.

The model theoretical meaning of a normal logic program is defined in the same manner as that of a Horn logic program with an additional statement that an interpretation I satisfies default negation $not\ A$ whenever A is not true in I ($I \models not\ A$ iff $I \not\models A$). Then a *model* of a normal logic program is defined as an interpretation satisfying every normal clause from the program. A normal logic program is *consistent* if it has a model, otherwise it is *inconsistent*.

A declarative semantics of normal logic programs is defined in terms of the particular Herbrand models called the *minimal models*. An Herbrand model M is a minimal model of a program P if no proper subset of M is a model of P.

In case of definite logic programs, every program has exactly one minimal model, the least Herbrand model. However, in the presence of non-Horn clauses, a program may have several minimal models in general. In this situation, some *canonical models* are usually chosen as the *intended meaning* of a program. By the "intended meaning" we mean that one who writes/reads the program is likely to expect it.

One of the criteria for choosing canonical models is the *supportedness* [Apt et al., 1988]. For a program P, a model M is called *supported* if for any atom A included in M, there is a ground clause of the form (2.3) from P such that $A = A_l$ and M satisfies the body of the clause.

The condition of supportedness is quite natural and appealing, since we are interested in a model which includes atoms actually derived from a program. For example, the program $\{\ a \leftarrow not\ b\ \}$ has two minimal models $\{a\}$ and $\{b\}$, where the first one is supported while the second one is not. In fact, b is never derived from the program and the intended meaning of the above program is $\{a\}$ reflecting the sentence that "a is true if b is not proved".

The *stable model semantics* proposed by Gelfond and Lifschitz [1988] characterizes such canonical models for normal logic programs.

Given a normal logic program P, an interpretation I is called a *stable model* of P if I coincides with the least Herbrand model of the Horn logic program P^I defined as

$$P^I = \{\ A_l \leftarrow B_1 \wedge \ldots \wedge B_m \mid \text{there is a ground clause of the form (2.3) from } P \text{ such that } \{B_{m+1}, \ldots, B_n\} \cap I = \emptyset\ \}.$$

An intuitive meaning of the above definition is that we first consider an *assumption set* I, and compute the *reduct* P^I of P with respect to the assumption set I. Then, if I coincides with the set of logical consequences of P^I, the assumption set I is justified as an intended meaning.

Stable models are minimal and supported models, but not vice versa [Marek and Subrahmanian, 1992]. A normal logic program may have none, one, or multiple stable models in general. In particular, a consistent (locally) stratified logic program has a unique stable model called the *perfect model* [Przymusinski, 1988a].

Note that a consistent program does not always have a stable model. For example, the program $\{\ a \leftarrow,\quad b \leftarrow not\ b\ \}$ is consistent since it has a model $\{a, b\}$, while the program has no stable model. A program having at least one stable model is called *coherent*, otherwise it is called *incoherent*. For sufficient conditions of the existence

of a stable model, call-consistent logic programs or order-consistent logic programs are always coherent [Fages, 1990; Dung, 1992b].

Another well-known semantics for normal logic programs is the *well-founded semantics*, originally proposed by van Gelder et al. [1991]. We do not address the detailed definition of the well-founded semantics here, but refer the reader to the literature [van Gelder et al., 1991; Przymusinski, 1989a; van Gelder, 1989].

The stable model semantics and the well-founded semantics provide alternative theoretical frameworks for normal logic programs. An essential difference between two formalisms is that the stable model semantics is based on the classical two-valued logic, while the well-founded semantics is based on the notion of *three-valued well-founded partial model*.

The well-founded semantics has an advantage that the well-founded (partial) model is uniquely defined for any consistent normal logic program, while this is not the case for the stable model semantics. However, this deterministic feature often makes the well-founded semantics too skeptical compared with the stable model semantics. For example, the program $\{\ c \leftarrow a, \quad c \leftarrow b, \quad a \leftarrow not\ b, \quad b \leftarrow not\ a\ \}$ has two stable models $\{a, c\}$ and $\{b, c\}$, while its well-founded model is the empty set (i.e. each atom has truth value *unknown*). Thus, the stable model semantics or the well-founded semantics is usually chosen according to applications.

Recent studies of logic programming have extended the stable model semantics and the well-founded semantics in various ways, while their original forms still serve as two representative semantics for normal logic programs at present. Moreover, both formalisms are revealed to be closely related with each other [Przymusinski, 1990c; Dung, 1992b; Baral and Subrahmanian, 1993].

For proof procedures, bottom-up algorithms are proposed for computing stable models in [Sacca and Zaniolo, 1990; Eshghi, 1990; Inoue et al., 1992; Fernandez et al., 1993; Subrahmanian et al., 1993; Bell et al., 1993], and for computing well-founded models in [Kemp et al., 1992; Subrahmanian et al., 1993].

As a top-down proof procedure for normal logic programs, SLD-resolution is extended to *SLDNF-resolution* [Clark, 1978; Lloyd, 1987]. For stratified logic programs, SLDNF-resolution is modified to cope with infinite failure in [Kemp and Topor, 1988; Seki and Itoh, 1988; Przymusinski, 1989c] under the perfect model semantics. An *abductive proof procedure* by Eshghi and Kowalski [1989] is a procedure based on SLDNF-resolution and is correct with respect to the stable model semantics for call-consistent logic programs. For the stable model semantics, however, there is no simple top-down proof procedure applicable for *any* normal logic program in general. This is due to the *irrelevance* property of the stable model semantics [Dix, 1992b] such that

the truth value of an atom is often affected by a clause irrelevant to its derivation. On the other hand, some general top-down proof procedures are known for the well-founded semantics such as [Ross, 1989a; Przymusinski, 1989a; Bidoit and Legay, 1990; Chen and Warren, 1993].

2.2.3 Extended Logic Programs

In logic programming, it is often useful to represent explicit negation as well as default negation in a program. Gelfond and Lifschitz [1990] have extended the framework of normal logic programs to include *classical negation* in a program.

An *extended logic program* is a finite set of *extended clauses* of the form:

$$L_l \leftarrow L_1 \wedge \ldots \wedge L_m \wedge not\ L_{m+1} \wedge \ldots \wedge not\ L_n \quad (0 \leq l \leq 1;\ n \geq m \geq 0) \qquad (2.4)$$

where each L_i is a literal. A *not*-free extended logic program is called a *positive extended logic program*. An extended clause with the empty head and a non-empty body is called an *integrity constraint*.

Note that in extended logic programs, two kinds of negation, classical negation ($\neg$) and default negation (*not*), are distinguished.[3] For example, awarding scholarships to college students is written by the following extended logic program [Gelfond and Lifschitz, 1990]:

$$Eligible(x) \leftarrow HighGPA(x),$$
$$\neg Eligible(x) \leftarrow \neg FairGPA(x),$$
$$Interview(x) \leftarrow not\ Eligible(x) \wedge not\ \neg Eligible(x),$$

where $\neg Eligible(x)$ means that x is non-eligible, while $not\ Eligible(x)$ means that there is no evidence that x is eligible.

Introduction of classical negation enables us to represent negative knowledge explicitly in a program. While positive and negative knowledge are now equivalently specified in a program, those not explicitly represented in a program are considered to be *unknown*. This means that the closed world assumption is not assumed in an extended logic program any more, but is specified for each predicate by the following *CWA-rule*:

$$\neg P(x) \leftarrow not\ P(x).$$

Thus, in an extended logic program we can freely specify the closed world assumption depending on whether a relation is complete or not. In this sense, extended logic programs can represent incomplete knowledge in a program.

[3] In contrast to default negation, classical negation is also called *explicit negation*.

The stable model semantics for normal logic programs is extended to the *answer set semantics* for extended logic programs [Gelfond and Lifschitz, 1990]. The answer sets are defined in two steps.

Let P be a positive extended logic program and $\mathcal{L}_P$ be the set of all ground literals from the language of P. Then, an *answer set* of P is defined as the smallest subset S of $\mathcal{L}_P$ satisfying the conditions:

1. For each ground clause $L_l \leftarrow L_1 \wedge \ldots \wedge L_m$ from P, $\{L_1, \ldots, L_m\} \subseteq S$ implies $L_l \in S$. In particular, for each ground clause $\leftarrow L_1 \wedge \ldots \wedge L_m$ from P, $\{L_1, \ldots, L_m\} \not\subseteq S$; and

2. If S contains a pair of complementary literals L and $\neg L$, then $S = \mathcal{L}_P$.

Next, let P be an extended logic program and $S \subseteq \mathcal{L}_P$. The *reduct* P^S of P with respect to S is defined as

$$P^S = \{\ L_l \leftarrow L_1 \wedge \ldots \wedge L_m \mid \text{there is a ground clause of the form (2.4) from } P \text{ such that } \{L_{m+1}, \ldots, L_n\} \cap S = \emptyset\ \}.$$

Then S is an answer set of P if S is an answer set of P^S. An extended logic program has either *consistent answer sets* different from $\mathcal{L}_P$, or the unique *contradictory answer set* $\mathcal{L}_P$, or no answer set. An extended logic program having at least one answer set is called *coherent*, otherwise it is called *incoherent*.

Gelfond and Lifschitz also show a syntactic translation from extended logic programs into normal logic programs. Given an extended logic program P, its *positive form* P^+ is defined as a normal logic program obtained by replacing each negative literal $\neg A$ in P with a newly introduced atom A' in P^+ where A' and A have the same arity. For example, the previous program is rewritten by the following positive form:

$$\begin{aligned} &Eligible(x) \leftarrow HighGPA(x), \\ &Eligible'(x) \leftarrow FairGPA'(x), \\ &Interview(x) \leftarrow not\ Eligible(x) \wedge not\ Eligible'(x). \end{aligned}$$

Then consistent answer sets of an extended logic program P are expressed in terms of stable models of the corresponding normal logic program P^+.

Note that the semantics of extended logic programs is different from classical first-order logic even in the absence of default negation in a program. For instance, the meanings of the clauses $L_1 \leftarrow L_2$ and $\neg L_2 \leftarrow \neg L_1$ are different in an extended logic program; given L_2, L_1 is derived from the first clause, but not from the second clause. In this sense, an extended clause is not contrapositive with respect to $\leftarrow$ and $\neg$.

The well-founded semantics is also generalized to extended logic programs in [Przymusinski, 1990a; Pereira and Alferes, 1992]. Logic programming with two kinds of negation is studied in [Wagner, 1991a; Alferes and Pereira, 1992] from different viewpoints. Proof procedures for extended logic programs are developed in [Inoue et al., 1992; Teusink, 1993b; Alferes et al., 1994].

As presented in this section, a framework of logic programming starts from classical first-order logic, while it becomes quite different from classical logic when negation is introduced in a program. In a normal/extended logic program, each clause has its intended meaning depending on its syntax (written form), and is viewed as a *derivation rule* or a *constructive statement* [Bry, 1989] rather than just a clause. With this reason, a clause in a program is often called a *rule* in some literature, but we abuse the term "clause" in this dissertation as far as no confusion arises.

2.3 Disjunctive Logic Programming

Logic programs introduced in the previous section specify knowledge having a definite consequence in the head of each clause. By contrast, logic programs possibly including clauses with disjunctive heads are called *indefinite logic programs* or *disjunctive logic programs* [Gallaire et al., 1984; Lobo et al., 1992]. Indefinite or disjunctive logic programs are more expressive than definite logic programs since they can represent indefinite information as well as definite one in a program. Our primary interest in this dissertation is in such programs, and hereafter we use the term disjunctive logic programs or simply disjunctive programs.

2.3.1 Positive Disjunctive Programs

A foundation of disjunctive programs is firstly studied by Minker [1982], in which he provided a theoretical framework of positive disjunctive programs.

A *positive disjunctive program* is a finite set of clauses of the form:

$$A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m \quad (l, m \geq 0) \tag{2.5}$$

where A_i's and B_j's are atoms. A clause is called *disjunctive* if its head contains more than one atom ($l > 1$); otherwise it is a Horn clause. Thus a positive disjunctive program is a generalization of a Horn logic program.

In a disjunctive program, indefinite information is specified using disjunctive clauses. That is, when the condition of a disjunctive clause (2.5) holds, at least one of the disjuncts A_i $(1 \leq i \leq l)$ becomes true, but it is generally unknown which one is true.

In real life situations, disjunctive knowledge appears in two ways. For example, the clause:

$$male \vee female \leftarrow human$$

is intended to mean an *exclusive* disjunction such that *human* is either *male* or *female* but not both. On the other hand, the clauses:

$$land\text{-}animal \vee acquatic \leftarrow animal\,,$$

$$amphibian \leftarrow land\text{-}animal \wedge acquatic$$

have an *inclusive* disjunction meaning that if an animal is both a *land-animal* and *acquatic*, it is an *amphibian*.

A declarative semantics of positive disjunctive programs is given by the *minimal model semantics* [Minker, 1982], which is defined as the set of all minimal models of a program. A *consistent* positive disjunctive program has at least one minimal model, while an *inconsistent* program has none. The minimal model semantics coincides with the least Herbrand model semantics in Horn logic programs. The minimal model semantics also has its origin in McCarthy's *circumscription* [McCarthy, 1980] and close relationships between circumscription and logic programming semantics are studied in [Gelfond et al., 1989].

Inference rules for default negation are also introduced in disjunctive programs. In the presence of disjunctive information in a program, however, Reiter [1978] pointed out that the CWA causes an inconsistency in a program. For example, in the program $\{a \vee b \leftarrow\}$, the CWA implies both $\neg a$ and $\neg b$, which are inconsistent with the program. To improve such a situation in disjunctive programs, two alternative extensions of the CWA are known. One is the *generalized closed world assumption* (GCWA) by Minker [1982], and the other is the *weak generalized closed world assumption* (WGCWA) by Rajasekar et al. [1989]. The WGCWA is also independently proposed by Ross and Topor [1988] under the name of the *disjunctive database rule* (DDR). Comparing each rule, the GCWA provides a strong inference rule for negation, while the WGCWA or the DDR provides a weak inference rule for negation. Both extensions fairly generalize the CWA, however, each rule has some limitation in its inference ability as discussed in Chapter 3. The completion semantics of normal logic programs is also extended to disjunctive programs in [Lobo et al., 1992].

Proof procedures for the minimal model semantics and the GCWA are studied in the literature such as [Bossu and Siegel, 1985; Yahya and Henschen, 1985; Grant and Minker, 1986; Henschen and Park, 1988; Manthey and Bry, 1988; Przymusinski, 1989b; Lobo et al., 1989; Minker and Rajasekar, 1990; Fernandez and Minker, 1991a], while those for the WGCWA are in [Ross and Topor, 1988; Rajasekar et al., 1989].

2.3.2 Normal Disjunctive Programs

As the case of Horn logic programs, positive disjunctive programs are extended to normal disjunctive programs by incorporating default negation in a program.

A *normal disjunctive program* is a finite set of clauses of the form:

$$A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not B_{m+1} \wedge \ldots \wedge not B_n \quad (l \geq 0, n \geq m \geq 0) \quad (2.6)$$

where A_i's and B_j's are atoms and *not* is the negation-as-failure operator. A clause is called a (normal) disjunctive clause if its head contains more than one atom; otherwise it is a normal clause. A normal disjunctive program reduces to a positive disjunctive program in the absence of default negation, and reduces to a normal logic program in the absence of disjunctive clauses.

The notions of models and consistent/inconsistent normal disjunctive programs are defined in the same way as those of normal logic programs. The stable model semantics of normal logic programs is directly extended to normal disjunctive programs as follows.

Given a normal disjunctive program P, an interpretation I is called a *(disjunctive) stable model* [4] of P if I coincides with a minimal model of the positive disjunctive program P^I (called a *reduct*) defined as

$$P^I = \{\, A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m \mid \text{there is a ground clause of the form (2.6) from } P \text{ such that } \{B_{m+1}, \ldots, B_n\} \cap I = \emptyset \,\}.$$

The *disjunctive stable model semantics* [Przymusinski, 1991a; Gelfond and Lifschitz, 1991] is defined as the set of all stable models of the program. By definition, the disjunctive stable model semantics reduces to Gelfond and Lifschitz's stable model semantics in normal logic programs, and also coincides with the perfect model semantics in *stratified disjunctive programs* [Przymusinski, 1988a].[5]

[4] We use the term disjunctive stable models when we distinguish them from stable models of normal logic programs. Otherwise, we simply say stable models.

[5] The definition of a stratified disjunctive program is a direct extension of that of a stratified logic program.

The notion of supported models is also extended to disjunctive programs. Given a normal disjunctive program P, a model M of P is *supported* if for any atom A included in M, there is a ground clause of the form (2.6) from P such that $A = A_i$ $(1 \leq i \leq l)$ and M satisfies the body of the clause. Stable models of a normal disjunctive program are minimal and supported models.

Like normal logic programs, there are consistent normal disjunctive programs having no stable model. A normal disjunctive program is called *coherent* (resp. *incoherent*) if it has a (resp. no) stable model.

The well-founded semantics of normal logic programs is also extended to the *disjunctive well-founded semantics* [Ross, 1989b; Baral et al., 1992a] or the *stationary semantics* [Przymusinski, 1990b; 1991b] in normal disjunctive programs. Proof procedures for normal disjunctive programs are developed in the literature such as [Ross, 1989b; Baral et al., 1992a; Inoue et al., 1992; Dung, 1992a; Dung, 1993; Fernandez and Minker, 1992].

Now we compare the stable model semantics approach and the well-founded semantics approach to normal disjunctive programs. Both the disjunctive stable model semantics and the disjunctive well-founded/stationary semantics generalize the minimal model semantics of positive disjunctive programs. However, there are essential differences between two approaches as follows.

1. The disjunctive stable model semantics characterizes non-deterministic behavior of a disjunctive program in terms of multiple stable models. On the other hand, the disjunctive well-founded/stationary semantics presents the meaning of a program by a set of ground disjunctions called a *state*, which is different from classical approaches based on Herbrand interpretations.

2. The stable model semantics is closely related to nonmonotonic reasoning in AI. However, the well-founded semantics is based on the three-valued logic, and in order to relate the well-founded semantics to nonmonotonic formalisms, three-valued extensions of the formalisms are needed [Przymusinski, 1991c].

3. The stable model semantics is also known to be useful to characterize abductive reasoning in AI, while the well-founded semantics is not usually used for abduction due to its deterministic feature.

Thus, approaches based on the stable model semantics still stay in classical two-valued logic and are useful to characterize commonsense reasoning in disjunctive programs. With these reasons, we take the stable model approach rather than the well-founded model approach in this dissertation.

2.3.3 Extended Disjunctive Programs

In disjunctive programs, classical negation is also included under the framework of extended disjunctive programs.

An *extended disjunctive program* is a finite set of clauses of the form:

$$L_1 \vee \ldots \vee L_l \leftarrow L_{l+1} \wedge \ldots \wedge L_m \wedge not\ L_{m+1} \wedge \ldots \wedge not\ L_n \quad (n \geq m \geq l \geq 0) \quad (2.7)$$

where each L_i is a literal. A clause is called an (extended) disjunctive clause if its head contains more than one literal; otherwise it is an extended clause. An extended disjunctive program containing no *not* is called a *positive extended disjunctive program*. An extended disjunctive program reduces to a normal disjunctive program when all L_i's are atoms, and in the absence of disjunctive clauses it reduces to an extended logic program.

The *answer set semantics* of extended logic programs is directly extended to extended disjunctive programs as follows.

Let P be a positive extended disjunctive program and $\mathcal{L}_P$ be the set of all ground literals from the language of P. Then, an *answer set* of P is defined as the minimal subset S of $\mathcal{L}_P$ satisfying the conditions:

1. For each ground clause $L_1 \vee \ldots \vee L_l \leftarrow L_{l+1} \wedge \ldots \wedge L_m$ from P, $\{L_{l+1}, \ldots, L_m\} \subseteq S$ implies $L_i \in S$ for some i $(1 \leq i \leq l)$. In particular, for each ground clause $\leftarrow L_1 \wedge \ldots \wedge L_m$ from P, $\{L_1, \ldots, L_m\} \not\subseteq S$; and

2. If S contains a pair of complementary literals L and $\neg L$, then $S = \mathcal{L}_P$.

Next, let P be an extended disjunctive program and $S \subseteq \mathcal{L}_P$. The *reduct* P^S of P with respect to S is defined as

$$P^S = \{\ L_1 \vee \ldots \vee L_l \leftarrow L_{l+1} \wedge \ldots \wedge L_m \mid \text{ there is a ground clause of the form (2.7) from } P \text{ such that } \{L_{m+1}, \ldots, L_n\} \cap S = \emptyset\ \}.$$

Then S is an *answer set* of P if S is an answer set of P^S. The notions of consistent/contradictory answer sets, and coherent/incoherent programs are correspondingly defined as in extended logic programs. A *positive form* of an extended disjunctive program is also defined in the same manner as that of an extended logic program.

Proof procedures for extended disjunctive programs are studied in [Inoue et al., 1992; Ben-Eliyahu and Dechter, 1992].

In extended disjunctive programs, incomplete information is expressed by disjunctions as well as classical negation. In this sense, extended disjunctive programs provide a fairly general framework for representing commonsense knowledge.

Note that in extended disjunctive programs, the non-contrapositive feature of extended clauses is inherited from extended logic programs. Moreover, the meaning of the connective $\vee$ is different from classical logic. For instance, the clause $a \vee \neg a \leftarrow$ is a tautology in its classical sense, then the programs $\{ b \leftarrow a, \quad a \vee \neg a \leftarrow \}$ and $\{ b \leftarrow a \}$ are equivalent under first-order logic. However, the first program has two answer sets $\{a, b\}$ and $\{\neg a\}$, while the second one has the empty answer set. This is because an answer set represents a set of literals possibly derived from a program, then in the absence of the clause $a \vee \neg a \leftarrow$, there is no way to derive a nor $\neg a$ in the second program. In this sense, the clause $a \vee \neg a \leftarrow$ is not a tautology any more. Hence, another connective "|" instead of $\vee$ is often used in some literature, but for notational convenience we use the classical connective $\vee$ in this dissertation.

2.4 Fixpoint Theory

Fixpoint theory describes denotational aspects of programming languages. In logic programming, it is usually used to characterize declarative semantics of logic programs. For a definite logic program, van Emden and Kowalski [1976] introduced a closure operator which acts over the lattice of Herbrand interpretations $2^{\mathcal{HB}_P}$.

Given a definite logic program P and its interpretation I, an *immediate consequence operator* T_P is defined as the mapping $T_P : 2^{\mathcal{HB}_P} \rightarrow 2^{\mathcal{HB}_P}$ such that

$$T_P(I) = \{ A \mid A \leftarrow B_1 \wedge \ldots \wedge B_m \text{ is a ground clause from } P \text{ and } \{B_1, \ldots, B_m\} \subseteq I \}.$$

The *ordinal powers* of T_P is defined as:

$$\begin{aligned} T_P \uparrow 0 &= \emptyset, \\ T_P \uparrow n+1 &= T_P(T_P \uparrow n), \\ T_P \uparrow \omega &= \bigcup_{n<\omega} T_P \uparrow n, \end{aligned}$$

where n is a successor ordinal and ω is a limit ordinal. Then *the least fixpoint* of T_P is given by $T_P \uparrow \omega$.

Van Emden and Kowalski have shown that the above least fixpoint coincides with the least Herbrand model of a definite logic program P. The result is extended by Apt, Blair and Walker [1988], in their paper they developed a nonmonotonic fixpoint

operator and its *iterative fixpoint* for logic programs with negation. Przymusinski [1988a] showed that Apt et al.'s fixpoint characterizes the perfect model of a stratified logic program. Further extensions of fixpoint semantics for normal logic programs are studied for the stable model semantics [Sacca and Zaniolo, 1990; Fages, 1991] and the well-founded semantics [van Gelder, 1989; Przymusinski, 1989a; van Gelder et al., 1991].

In the context of disjunctive programs, Minker and Rajasekar [1990] introduced a new fixpoint semantics for positive disjunctive programs. They first introduced the *extended Herbrand base* as the set of all ground positive disjunctions from a program, and defined the notion of a *state* as a set of positive disjunctions from the extended base. Then they developed a fixpoint operator which operates on states and showed that its least fixpoint contains disjunctions which are true in every minimal model of a program.

Such a state based fixpoint semantics is extended to stratified disjunctive programs [Rajasekar and Minker, 1989] and normal disjunctive programs [Baral et al., 1992a; Przymusinski, 1990b; Przymusinski, 1991b]. Yet other fixpoint semantics are also presented in the literature such as [Ross and Topor, 1988; Reed et al., 1991; Fernandez and Minker, 1991b; Decker, 1992; Inoue and Sakama, 1992]. Detailed discussion on the fixpoint semantics of disjunctive programs is presented in Chapter 3.

2.5 Computational Complexity

In this section, we briefly review basic concepts of complexity theory. More on the snbject can be found in [Garey and Johnson, 1979; Johnson, 1990].

Complexity theory is used to classify problems according to their intrinsic computational difficulties. For classification, some complexity classes are introduced to characterize problems. In this dissertation, we are mainly concerned with *decision problems* which admit boolean answers (yes/no).

The class P represents the set of all decision problems solvable hy *deterministic Turing machines* in polynomial time. A problem in this class is called *tractable* or *efficiently* solvable.

On the other hand, the class NP represents the set of all decision problems solvable by *non-deterministic Turing machines* in polynomial time. The class co-NP is the set of problems whose complements are in NP. Apparently, the class P is included both in the class NP and in co-NP. The question whether this inclusion is strict remains open, but it is usually considered that P$\neq$NP.

The *polynomial hierarchy* consists of classes Δ_k^P, Σ_k^P, and Π_k^P defined as follows:

$$\Delta_0^P = \Sigma_0^P = \Pi_0^P = \mathrm{P},$$

$$\Delta_{k+1}^P = \mathrm{P}^{\Sigma_k^P}, \quad \Sigma_{k+1}^P = \mathrm{NP}^{\Sigma_k^P}, \quad \Pi_{k+1}^P = \text{co-}\Sigma_{k+1}^P \ \ (k \geq 0).$$

In particular, Δ_1^P=P, Σ_1^P=NP, and Π_1^P=co-NP.

In the above, Δ_{k+1}^P (resp. Σ_{k+1}^P) is the set of problems solvable deterministically (resp. non-deterministically) in polynomial time with an *oracle* for the problems in Σ_k^P, while Π_{k+1}^P consists of the complements of Σ_{k+1}^P. An oracle is considered as a subroutine without cost, that is, it is decided in unit time for any problem.

In the polynomial hierarchy, the relations $\Delta_k^P \subseteq \Sigma_k^P \cap \Pi_k^P$ and $\Sigma_k^P \cup \Pi_k^P \subseteq \Delta_{k+1}^P$ hold. Problems included in the upper levels of the hierarchy are more difficult to solve than those in lower levels unless the polynomial hierarchy collapses.

A problem is called *C-hard* if every problem in a class C is efficiently reducible to the problem by a polynomial-time transformation. The C-hard problem is also called *C-complete* if the problem itself belongs to the class C. For instance, checking the satisfaction of any propositional theory is an NP-complete problem.

When we discuss the complexity issue of logic programming, (finite) propositional logic programs are usually assumed. This is because in the presence of function symbols, a program containing variables has the infinite set of ground clauses, then checking its satisfiability is undecidable even in the case of Horn logic programs.[6] For propositional Horn logic programs, deciding whether a ground atom is derived from a program is tractable. This is also the case for the decision problem under the perfect model semantics of propositional stratified logic programs [Apt et al., 1988].

On the other hand, when a propositional logic program has multiple canonical models, the computational complexity of the program is usually characterized by the following three decision problems.

Existence: Deciding the existence of a canonical model of a program.

Set-Membership: Deciding whether a ground atom is true in some canonical model of a program.

Set-Entailment: Deciding whether a ground atom is true in every canonical model of a program.

[6]For programs containing variables, their undecidabilities are characterized by the *arithmetic hierarchy* of recursion theory. But we do not discuss such recursion-theoretic characterizations in the dissertation.

The set-membership problem also characterizes *credulous reasoning*, while the set-entailment problem characterizes *skeptical reasoning*. For example, the existence problem and the set-membership problem for the stable model semantics of normal logic programs are both NP-complete, while the set-entailment problem is coNP-complete [Marek and Truszczynski, 1991a; 1991b]. In disjunctive programs, problems become harder than normal logic programs in general. The set-membership problem for the disjunctive stable model semantics is Σ_2^P-complete, while the set-entailment problem is Π_2^P-complete. It is known that similar results also hold for nonmonotonic logics [Gottlob, 1992].

For surveys on the complexity issues in logic programming, see [Schlipf, 1992b; Eiter and Gottlob, 1993a; Cadori and Schaerf, 1993].

Chapter 3

Possible Model Semantics for Disjunctive Logic Programs

In this chapter, we introduce a new declarative semantics of disjunctive programs called the *possible model semantics*. The possible model semantics is an alternative theoretical framework to the classical minimal model semantics and provides a flexible inference mechanism for inferring negation in disjunctive programs. The possible model semantics is characterized by a new fixpoint semantics of disjunctive programs. We also present a proof procedure for the possible model semantics and show that the possible model semantics has an advantage from the computational complexity point of view.

3.1 Introduction

Traditionally, the declarative semantics of logic programming and deductive databases has been studied based on the notion of *minimal models*. For instance, the *least Herbrand model semantics* for Horn logic programs [van Emden and Kowalski, 1976], the *prefect model semantics* for stratified logic programs [Przymusinski, 1988a], and the *stable model semantics* for normal logic programs [Gelfond and Lifschitz, 1988] are all minimal models. The minimal models reflect the so-called *Occam's razor* such that "only those objects should be assumed to exist which are minimally required by the context". Such a *principle of minimality* plays a fundamental role in the area of not only logic programming but also *nonmonotonic reasoning* in artificial intelligence [McCarthy, 1980]. Therefore, it has been recognized that the principle of minimality is one of the most basic and indispensable criteria that each semantics for commonsense reasoning should obey [Schlipf, 1992a].

This is also the case in the context of disjunctive programs, namely, the *minimal model semantics* for positive disjunctive programs [Minker, 1982] and the *disjunctive stable model semantics* for normal disjunctive programs [Przymusinski, 1988b; Gelfond and Lifschitz, 1991] are both minimal. However, it has also been pointed out that such a minimalism is not always appropriate in a theory containing indefinite information. Ross and Topor [1988] have firstly noticed this problem in the context of inferring negation in disjunctive programs. They argue that when one infers negation from a disjunctive program, one should be cautious to interpret disjunctions *inclusively* rather than *exclusively*. In fact, the minimal model semantics minimizes truth extensions of predicates as much as possible, then it usually interprets disjunctions exclusively and maximizes negative information inferred from a program.

In positive disjunctive programs, Minker [1982] has extended Reiter's CWA to the *generalized closed world assumption* (GCWA). On the other hand, Ross and Topor [1988] have proposed an alternative rule called the *disjunctive database rule* (DDR), which turns out to be equivalent to the *weak generalized closed world assumption* (WGCWA) independently proposed by Rajasekar et al. [1989].

Comparing these two rules, the GCWA is based on the minimal model semantics and usually interprets disjunctions exclusively, while the DDR and the WGCWA are weaker than the GCWA and interpret disjunctions inclusively. Thus, both the GCWA and the WGCWA (or DDR) fairly extend the CWA, but the problem is that they are inherently *different* from each other. In fact, in the absence of a single uniform framework, one has to use these separate rules in order to treat both exclusive and inclusive disjunctions in the same program. Such a situation actually happens in our real life situations. For instance, consider the program:

$$land\text{-}animal \vee aquatic \leftarrow animal,$$

$$biped \vee quadruped \leftarrow land\text{-}animal,$$

$$amphibian \leftarrow land\text{-}animal \wedge aquatic,$$

in which the disjunction in the first clause is inclusive, while the disjunction in the second clause is exclusive. As another example, a disjunctive clause possibly contains *hybrid* disjunctions such as:

$$Sunday \vee national\text{-}holiday \vee weekday \leftarrow calendar\text{-}days,$$

in which $Sunday \vee weekday$ is exclusive, while $Sunday \vee national\text{-}holiday$ is inclusive.

To treat such kinds of programs, we need a single framework which can distinguish two kinds of disjunctions. Another point is that Ross and Topor, and Rajasekar et al.

have provided a rule for inferring negation in inclusive disjunctive programs, however, they concern only negative information in a program and do not provide any model theoretical meaning for inclusive disjunctive programs as a counterpart of the minimal model semantics.

In this chapter, we present an alternative approach to the declarative semantics of disjunctive programs. We first introduce a new semantics called the *possible model semantics* for disjunctive programs. In contrast to the classical minimal model semantics, the possible model semantics considers not only minimal models but also certain kinds of non-minimal models in a program. Due to its non-minimal feature, we show that the possible model semantics enjoys several interesting properties. Especially by treating both inclusive and exclusive disjunctions uniformly in a program, it can provide a flexible mechanism for inferring negation in a program. Next we present a new fixpoint semantics of disjunctive programs. We introduce a mapping operating over sets of interpretations and show that its fixpoint closure characterizes the possible models of a disjunctive program. We also develop an algorithm to compute the possible model semantics in normal disjunctive programs which is based on a bottom-up model generation proof procedure. We finally discuss the computational complexity of the possible model semantics and show that the possible model semantics has a computational advantage compared with other minimal model based semantics.

The rest of this chapter is organized as follows. In Section 3.2, we introduce the possible model semantics for positive disjunctive programs and present its properties. The possible model semantics is also extended to normal disjunctive programs in Section 3.3. In Section 3.4, we propose a new fixpoint semantics of disjunctive programs to characterize the possible model semantics. A proof procedure for computing possible models is given in Section 3.5. In Section 3.6, we discuss the computational aspect of the possible model semantics. Section 3.7 presents detailed comparisons with related work, and Section 3.8 summarizes this chapter.

3.2 Possible Model Semantics for Positive Disjunctive Programs

In this section, we first consider positive disjunctive programs, that is, disjunctive programs containing no default negation.

3.2.1 Negation in Positive Disjunctive Programs

As presented in Section 2.3, Reiter's CWA is not useful in the context of disjunctive programs. Then, for inferring negation in positive disjunctive programs, Reiter's CWA is mainly extended in two ways: one is Minker's generalized closed world assumption (GCWA) and the other is Ross and Topor's disjunctive database rule (DDR) or Rajasekar et al.'s weak generalized closed world assumption (WGCWA).[1] We first review definitions and properties of those two frameworks.

Given a positive disjunctive program P, let us denote by $\mathcal{MM}_P$ the set of all minimal models of P. Then the GCWA is defined as follows.

Definition 3.1 ([Minker, 1982]) Let P be a consistent positive disjunctive program. Then $GCWA(P)$ is defined as:

$$GCWA(P) = \{\neg A \mid A \in \mathcal{HB}_P \text{ and } A \notin I \text{ for any } I \in \mathcal{MM}_P\}. \quad \square$$

On the other hand, the WGCWA provides a weaker form of closed world reasoning in positive disjunctive programs as follows.

The *Horn transformation* [Ross and Topor, 1988] of a positive disjunctive program P is defined as:

$$Horn(P) = \{A_i \leftarrow B_1 \wedge \ldots \wedge B_m \mid A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m \in P \text{ and } 1 \leq i \leq l,\ l \geq 1\}.$$

Note here that $Horn(P)$ is always consistent since it does not contain integrity constraints.

Definition 3.2 ([Ross and Topor, 1988; Rajasekar et al., 1989])
Let P be a consistent positive disjunctive program and $Horn(P)$ be its Horn transformation. Let $M_{Horn(P)}$ be the least Herbrand model of $Horn(P)$. Then $WGCWA(P)$ is defined as:

$$WGCWA(P) = \{\neg A \mid A \in \mathcal{HB}_P \text{ and } A \notin M_{Horn(P)}\}. \quad \square$$

Properties of the GCWA and the WGCWA are as follows.

Theorem 3.1 ([Minker, 1982; Ross and Topor, 1988; Rajasekar et al., 1989])
Let P be a consistent positive disjunctive program and A be a ground atom. Then,

[1] According to [Rajasekar et al., 1989; Lobo et al., 1992], the DDR and the WGCWA are equivalent. Then we use the term WGCWA hereafter.

(i) $P \cup GCWA(P)$ is consistent.
$P \cup WGCWA(P)$ is consistent.

(ii) $P \models A$ iff $P \cup GCWA(P) \models A$.
$P \models A$ iff $P \cup WGCWA(P) \models A$.

(iii) $P \subseteq P'$ does not imply $GCWA(P') \subseteq GCWA(P)$.
$P \subseteq P'$ implies $WGCWA(P') \subseteq WGCWA(P)$.

(iv) $WGCWA(P) \subseteq GCWA(P)$.

(v) For a definite logic program P, $GCWA(P) = WGCWA(P) = CWA(P)$. $\square$

That is, (i) both $GCWA(P)$ and $WGCWA(P)$ are *consistent* with P, (ii) positive facts proven from P are *invariant*, (iii) the GCWA (resp. WGCWA) is *non-decreasing* (resp. *decreasing*), (iv) the GCWA is *stronger* than the WGCWA, and (v) for definite logic programs each rule *reduces* to the CWA.

Example 3.1 Let P be the program:

$$\{\, a \vee b \vee c \leftarrow,\quad d \leftarrow a \wedge b,\quad e \leftarrow b \wedge c,\quad \leftarrow b \wedge c \,\}$$

where $\mathcal{MM}_P = \{\{a\}, \{b\}, \{c\}\}$ and $M_{Horn(P)} = \{a, b, c, d, e\}$. Then, $GCWA(P) = \{\neg d, \neg e\}$, while $WGCWA(P) = \emptyset$. $\square$

In the above example, the GCWA interprets each disjunction *exclusively*, while the WGCWA interprets them *inclusively*. Then, $GCWA(P)$ excludes the inclusive interpretation of $a \vee b$ and infers $\neg d$ from the program. On the other hand, the integrity constraint $\leftarrow b \wedge c$ inhibits an inclusive interpretation of $b \vee c$, nevertheless, $WGCWA(P)$ cannot infer $\neg e$. This is because the WGCWA does not consider the model theoretical meaning of a given program, and ignores the effect of integrity constraints in a program. In fact, $M_{Horn(P)}$ is no longer a model of P. Generally speaking, the GCWA is too strong to interpret inclusive disjunctions, while the WGCWA is too weak to treat exclusive disjunctions. Then, to treat both types of disjunctions in a program, one has to use two different rules in the same program.

To improve such a situation, in the next section we introduce a new declarative semantics called the *possible model semantics*, which can distinguish two types of disjunctions uniformly in a program.

3.2.2 Possible Model Semantics

A disjunctive program is considered to represent a set of possible facts that might have been true in the actual world. The possible model semantics is intended to formulate this situation.

Given a positive disjunctive program P, a *split program* is defined as a ground Horn logic program obtained from P by replacing each ground disjunctive clause of the form:

$$A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m$$

with the following ground definite clauses (called *split clauses*):

$$A_i \leftarrow B_1 \wedge \ldots \wedge B_m \quad \text{for every } A_i \in S$$

where S is some non-empty subset of $\{A_1, \ldots, A_l\}$. Note that every ground Horn clause from P is included in any split program of P. Then, a *possible model* of P is defined as the least Herbrand model of any split program of P. The set of all possible models of P is denoted by $\mathcal{PM}_P$.

Example 3.2 Let P be the program:

$$\{\, a \vee b \leftarrow, \quad b \vee c \leftarrow, \quad \leftarrow b \wedge c \,\}.$$

Then the split programs of P are

$$\{a \leftarrow, \quad b \leftarrow, \quad \leftarrow b \wedge c\},$$
$$\{a \leftarrow, \quad c \leftarrow, \quad \leftarrow b \wedge c\},$$
$$\{b \leftarrow, \quad \leftarrow b \wedge c\},$$
$$\{b \leftarrow, \quad c \leftarrow, \quad \leftarrow b \wedge c\},$$
$$\{a \leftarrow, \quad b \leftarrow, \quad c \leftarrow, \quad \leftarrow b \wedge c\}.$$

Since the last two split programs are inconsistent, the set of all possible models of P is $\mathcal{PM}_P = \{\{a, b\}, \{a, c\}, \{b\}\}$. □

Possible models have the following properties.

Proposition 3.2 A consistent positive disjunctive program has at least one possible model.

> *Proof:* Since a consistent positive disjunctive program has a consistent split program, the result immediately follows. □

Proposition 3.3 A possible model of a positive disjunctive program P is a model of P.

> *Proof:* Let M be a possible model of P such that M is the least Herbrand model of a consistent split program P_s. Then, for each clause $C' : A_i \leftarrow B_1 \wedge \ldots \wedge B_m$ in P_s, there is a corresponding clause $C : A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m$ in P where $1 \leq i \leq l$. Since M satisfies each C', it also satisfies C. Also each integrity constraint in P is included in P_s and is satisfied in M. Hence M is a model of P. □

The notion of possible models is different from minimal models. In fact, in Example 3.2, $\{a, b\}$ is a possible model, but not a minimal model. The intuitive meaning of the possible model is that each atom included in a possible model has its possible justification in a program. Thus, both inclusive and exclusive interpretations of disjunctions are considered whenever there is no integrity constraint to inhibit inclusive interpretations. The following property directly follows from the definition.

Proposition 3.4 A possible model is a supported model. □

Note that the converse of the above proposition does not hold in general. For example, $\{a\}$ is a supported model of the program $\{\, a \vee b \leftarrow a \,\}$, but not a possible model. The following relationship holds between possible models and minimal models of a program.

Proposition 3.5 Let P be a consistent positive disjunctive program. Then the set of all minimal elements from $\mathcal{PM}_P$ coincides with $\mathcal{MM}_P$.

> *Proof:* Let M be a minimal model of P. Then, for each clause $A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m$ in P, $\{B_1, \ldots, B_m\} \subseteq M$ implies $A_i \in M$ for some i $(1 \leq i \leq l)$. In this case, there is a split program P_s of P which contains $A_i \leftarrow B_1 \wedge \ldots \wedge B_m$. Since M satisfies each integrity constraint in P, M is the least Herbrand model of the consistent Horn logic program P_s, hence a possible model of P. Thus, $\mathcal{MM}_P$ is included in $\mathcal{PM}_P$. On the other hand, since each possible model is a model of P by Proposition 3.3, the set of all minimal elements from $\mathcal{PM}_P$ coincides with $\mathcal{MM}_P$. □

Thus the set of minimal models is a subset of the set of possible models. In particular, a definite logic program has a unique possible model which is the least Herbrand model of the program. The above proposition implies that as for the inference of positive facts, the possible model semantics coincides with the minimal model semantics.

Theorem 3.6 Let P be a consistent positive disjunctive program. An atom A is true in P iff A is included in every possible model of P. □

Note that possible models depend on the syntax of a program. For instance, two programs $\{ a \vee b \leftarrow, \quad a \leftarrow \}$ and $\{ a \leftarrow \}$ are equivalent under first-order logic, while the first program has the possible model $\{a, b\}$ which is not a possible model of the second program. This is because the first program is intended to specify some indefinite information about b, while it is not the case in the second program. Such a distinction is in fact meaningful in knowledge representation. Suppose a situation like that:

> There is a visitor at my house whom I do not know. I am living with my parents so that I guess either my mother or farther must know him: $know(mother, visitor) \vee know(father, visitor)$. After a while, mother comes back and she actually knows him: $know(mother, visitor)$, and at this moment the possibility is open that my father's knowing him too. However, if we replace the previous belief $know(mother, visitor) \vee know(father, visitor)$ with $know(mother, visitor)$, the negation $\neg know(father, visitor)$ is assumed under the closed world assumption.

Thus, logically equivalent sentences do not necessarily have the same meaning from the viewpoint of knowledge representation. Generally speaking, the syntax of a program plays an important role in logic programming to specify our intended knowledge, then it appears natural that the possible model semantics also shares such syntax-dependent properties.

Next we consider the inference of negative facts under the possible model semantics. Under the possible model semantics, negation is defined as follows.

Definition 3.3 Let P be a consistent positive disjunctive program. Then the *possible world assumption (PWA)* of P is defined as:

$$PWA(P) = \{\neg A \mid A \in \mathcal{HB}_P \text{ and } A \notin I \text{ for any } I \in \mathcal{PM}_P\}. \quad \square$$

Theorem 3.7 Let P be a consistent positive disjunctive program and A be a ground atom. Then,

(i) $P \cup PWA(P)$ is consistent.

(ii) $P \models A$ iff $P \cup PWA(P) \models A$.

(iii) $P \subseteq P'$ does not imply $PWA(P') \subseteq PWA(P)$.

(iv) For a definite logic program P, $PWA(P) = CWA(P)$.

Proof: (i) Since P is consistent, it has a minimal model M which is also a possible model of P. Then, by definition, M is also a model of $P \cup PWA(P)$, hence the result follows. (ii) The result directly follows from Theorem 3.6. (iii) Let $P = \{ a \vee b \leftarrow, \quad c \leftarrow a \wedge b \}$ and $P' = P \cup \{ \leftarrow a \wedge b \}$. Then $PWA(P) = \emptyset$, while $PWA(P') = \{\neg c\}$. (iv) For a definite logic program P, $\mathcal{PM}_P$ contains the unique least Herbrand model of P, hence the result follows. □

The next theorem presents that the PWA is stronger than the WGCWA and weaker than the GCWA.

Theorem 3.8 Let P be a consistent positive disjunctive program. Then the following relationship holds:

$$WGCWA(P) \subseteq PWA(P) \subseteq GCWA(P).$$

In particular, $WGCWA(P) = PWA(P)$ if $P \cup Horn(P)$ is consistent.

Proof: The relationship $PWA(P) \subseteq GCWA(P)$ immediately follows from the fact that $\mathcal{MM}_P \subseteq \mathcal{PM}_P$. Since the least Herbrand model $M_{Horn(P)}$ of $Horn(P)$ is a superset of any possible model in $\mathcal{PM}_P$, the relationship $WGCWA(P) \subseteq PWA(P)$ also holds. In particular, $Horn(P)$ is the set of all definite clauses included in the maximal split program of P. Then, if $P \cup Horn(P)$ is consistent, $M_{Horn(P)}$ coincides with the maximal element in $\mathcal{PM}_P$, hence the result follows. □

Example 3.3 (cont. from Example 3.1) Let P be the program:

$$\{ a \vee b \vee c \leftarrow, \quad d \leftarrow a \wedge b, \quad e \leftarrow b \wedge c, \quad \leftarrow b \wedge c \}$$

where $\mathcal{PM}_P = \{\{a\}, \{b\}, \{c\}, \{a, b, d\}, \{a, c\}\}$. Then, $PWA(P)$ implies $\neg e$, but not $\neg d$. □

Note that in the above example $P \cup Horn(P)$ is inconsistent and $WGCWA(P)$ fails to capture the intended meaning of P. By contrast, the possible model semantics can infer proper negation by distinguishing two kinds of disjunctions using integrity constraints.

The possible model semantics was independently discovered by Chan [1993] under the name of the *possible world semantics*. In [Chan, 1993], both notions are proven to be equivalent in positive disjunctive programs.

3.3 Possible Model Semantics for Normal Disjunctive Programs

In this section, we extend the possible model semantics of positive disjunctive programs to normal disjunctive programs.

The notion of split programs is defined in the same manner as positive disjunctive programs. Given a normal disjunctive program P, its *split program* is defined as a ground normal logic program obtained from P by replacing each ground disjunctive clause of the form:

$$A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\ B_{m+1} \wedge \ldots \wedge not\ B_n$$

with the following ground normal clauses (called *split clauses*):

$$A_i \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\ B_{m+1} \wedge \ldots \wedge not\ B_n \quad \text{for every } A_i \in S$$

where S is some non-empty subset of $\{A_1, \ldots, A_l\}$. Then, a *possible model* of P is defined as a stable model of any split program of P. The set of all possible models of P is denoted by $\mathcal{PM}_P$. A normal disjunctive program having at least one possible model is called *p-coherent*, otherwise it is called *p-incoherent*.

The possible models defined above reduce to those presented in the previous section in a positive disjunctive program. Also, possible models coincide with stable models in normal logic programs. The following properties hold.

Proposition 3.9 A possible model of a normal disjunctive program P is a model of P. □

Proposition 3.10 A possible model is a supported model, but not vice versa. □

Proposition 3.11 Let P be a consistent normal disjunctive program and $\mathcal{ST}_P$ be the set of all stable models of P. Then the set of all minimal elements from $\mathcal{PM}_P$ contains $\mathcal{ST}_P$.

Proof: By definition, a stable model M of P is also a stable model of some split program of P. Then M is also a possible model of P. Since M is minimal, it is also a minimal element in $\mathcal{PM}_P$. □

The above proposition implies that a coherent normal disjunctive program is also p-coherent (but not vice versa, see Example 3.9). The converse of the above proposition does not hold in general. That is, minimal possible models are not always stable models.

Example 3.4 Let P be the program:

$$\{\ a \vee b \leftarrow,\quad b \leftarrow a,\quad c \leftarrow not\ a\ \}.$$

Then $\mathcal{PM}_P = \{\{a,b\},\{b,c\}\}$, and $\{a,b\}$ is a minimal element in $\mathcal{PM}_P$ but not a stable model of P. □

In the above example, $\{b,c\}$ is the unique stable model of P, hence c is true under the disjunctive stable model semantics. However, this is not the case under the possible model semantics, since there is an inclusive interpretation of the disjunction $\{a,b\}$ in which c is not true. Thus, in contrast to the case of positive disjunctive programs, positive facts true under the possible model semantics (viz. positive facts true in every possible model) differ from those ones under the disjunctive stable model semantics. At the end of this subsection, we will also show the case that the possible model semantics implies more positive facts than the disjunctive stable model semantics.

Now we consider negative inference in normal disjunctive programs. We first define an extension of the GCWA as negation under the disjunctive stable model semantics.

Definition 3.4 Let P be a coherent normal disjunctive program. Then $GCWA^{\neg}(P)$ is defined as:

$$GCWA^{\neg}(P) = \{\neg A \mid A \in \mathcal{HB}_P \text{ and } A \notin I \text{ for any } I \in \mathcal{ST}_P\}.$$ □

Next, in order to define a suitable extension of the WGCWA, we introduce a transformation from a normal disjunctive program to a normal logic program.

The *NLP-transformation* of a normal disjunctive program P is defined as:

$$NLP(P) = \{A_i \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\ B_{m+1} \wedge \ldots \wedge not\ B_n \mid A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\ B_{m+1} \wedge \ldots \wedge not\ B_n \in P \text{ and } 1 \leq i \leq l,\ l \geq 1\}.$$

The $NLP(P)$ is a direct extension of $Horn(P)$, and $NLP(P) = Horn(P)$ holds for a positive disjunctive program P. For a coherent normal disjunctive program P, $NLP(P)$ is not always coherent.

Example 3.5 Let P be the program:

$$\{\ a \vee b \leftarrow not\ a\ \}.$$

Then its NLP-transformation becomes

$$NLP(P) = \{\ a \leftarrow not\ a,\quad b \leftarrow not\ a\ \}.$$

Here $\mathcal{ST}_P = \{\{b\}\}$, while $\mathcal{ST}_{NLP(P)} = \emptyset$. □

Conversely, there is an incoherent program whose NLP-transformation has a stable model.

Example 3.6 Let P be the program:

$$\{\ a \vee b \leftarrow,\quad b \leftarrow a,\quad \leftarrow not\ a\ \}.$$

Then $\mathcal{ST}_P = \emptyset$, while $\mathcal{ST}_{NLP(P)} = \{\{a, b\}\}$. □

We say that a normal disjunctive program P is *weakly coherent* if either P or $NLP(P)$ has a stable model. Clearly, a coherent program is also weakly coherent, but not vice versa.

Definition 3.5 Let P be a weakly coherent normal disjunctive program. Then $WGCWA^{\neg}(P)$ is defined as:

$$WGCWA^{\neg}(P) = \{\neg A \mid A \in \mathcal{HB}_P \text{ and } A \notin I \text{ for any } I \in \mathcal{ST}_P \cup \mathcal{ST}_{NLP(P)}\}\ \square$$

This definition is natural in the sense that $WGCWA^{\neg}(P)$ restricts its negative inference like the WGCWA by taking into account the stable models of $NLP(P)$. A similar extension is also given in [Dix, 1992a] in a different context. Note that in the above definition, one may consider that by analogy with the WGCWA, considering $\mathcal{ST}_{NLP(P)}$ is enough instead of $\mathcal{ST}_P \cup \mathcal{ST}_{NLP(P)}$. But this is not the case. For instance, let $P = \{\ a \vee b \leftarrow,\quad c \leftarrow not\ a,\quad c \leftarrow not\ b\ \}$. Then $\mathcal{ST}_P = \{\{a, c\}, \{b, c\}\}$, hence $GCWA^{\neg}(P)$ does not imply $\neg c$. On the other hand, $NLP(P) = \{\ a \leftarrow,\quad b \leftarrow,\ c \leftarrow not\ a,\quad c \leftarrow not\ b\ \}$, then $\mathcal{ST}_{NLP(P)} = \{\{a, b\}\}$ which implies $\neg c$. Thus, without $\mathcal{ST}_P$, the WGCWA$^{\neg}$ is not weaker than the GCWA$^{\neg}$ any more.

The PWA is also extended in normal disjunctive programs as follows.

Definition 3.6 Let P be a p-coherent normal disjunctive program. Then $PWA^{\neg}(P)$ is defined as:

$$PWA^{\neg}(P) = \{\neg A \mid A \in \mathcal{HB}_P \text{ and } A \notin I \text{ for any } I \in \mathcal{PM}_P\}.\ \square$$

Now we investigate the properties of each rule. In the following, we write $P \models_{ST} A$ (resp. $P \models_{PM} A$) if $A \in I$ for any $I \in \mathcal{ST}_P$ (resp. $I \in \mathcal{PM}_P$).

Theorem 3.12 Let P be a normal disjunctive program and A be a ground atom. Then the following properties hold.

1. (i) If P is coherent, $P \cup GCWA^{\neg}(P)$ is coherent.
   (ii) $P \models_{ST} A$ iff $P \cup GCWA^{\neg}(P) \models_{ST} A$.
   (iii) $P \subseteq P'$ does not imply $GCWA^{\neg}(P') \subseteq GCWA^{\neg}(P)$.
   (iv) For a positive disjunctive program P, $GCWA^{\neg}(P) = GCWA(P)$.
2. (i) If P is weakly coherent, $P \cup WGCWA^{\neg}(P)$ is weakly coherent.
   (ii) $P \models_{ST} A$ iff $P \cup WGCWA^{\neg}(P) \models_{ST} A$.
   (iii) $P \subseteq P'$ does not imply $WGCWA^{\neg}(P') \subseteq WGCWA^{\neg}(P)$.
   (iv) For a positive disjunctive program P, $WGCWA^{\neg}(P) = WGCWA(P)$.
3. (i) If P is p-coherent, $P \cup PWA^{\neg}(P)$ is p-coherent.
   (ii) $P \models_{PM} A$ iff $P \cup PWA^{\neg}(P) \models_{PM} A$.
   (iii) $P \subseteq P'$ does not imply $PWA^{\neg}(P') \subseteq PWA^{\neg}(P)$.
   (iv) For a positive disjunctive program P, $PWA^{\neg}(P) = PWA(P)$.

Proof: 1. (i) Since P is coherent, it has at least one stable model and every negated atom in $GCWA^{\neg}(P)$ is not included in any stable model of P, hence $P \cup GCWA^{\neg}(P)$ is coherent. (ii) Any negated atom in $GCWA^{\neg}(P)$ is not included in any stable model. Then adding such negative facts to P does not affect the construction of stable models. Hence the result follows. (iii) The GCWA$^{\neg}$ is non-decreasing since the GCWA$^{\neg}$ includes the GCWA (by (iv)) which is non-decreasing. (iv) Since stable models reduce to minimal models in a positive disjunctive program, the result immediately follows.

2. (i) When P is weakly coherent, every negated atom in $WGCWA^{\neg}(P)$ is not included in any stable model of P and $NLP(P)$. Hence, $P \cup WGCWA^{\neg}(P)$ is also weakly coherent. (ii) The result follows from (i). (iii) For the non-decreasing property of $WGCWA^{\neg}(P)$, see Example 3.7. (iv) Since $\mathcal{ST}_P \cup \mathcal{ST}_{NLP(P)}$ reduces to $\mathcal{MM}_P \cup \{M_{Horn(P)}\}$ in a positive disjunctive program P, and each minimal model in $\mathcal{MM}_P$ is a subset of $M_{Horn(P)}$, the result also holds.

3. (i) and (ii) follow directly from the definition. (iii) The PWA$^{\neg}$ is non-decreasing since the PWA$^{\neg}$ includes the PWA (by (iv)) which is non-decreasing. (iv) Since possible models in a normal disjunctive program reduce to those presented in the previous section in a positive disjunctive program, the result immediately follows. □

Notice that in contrast to the WGCWA, the WGCWA$^\neg$ is non-decreasing.

Example 3.7 Let P be the program:

$$\{\ a \vee b \leftarrow not\, c,\quad c \leftarrow d\ \}$$

and $P' = P \cup \{\ d \leftarrow\ \}$. Then, $\mathcal{ST}_P \cup \mathcal{ST}_{NLP(P)} = \{\{a\},\{b\},\{a,b\}\}$ and $\mathcal{ST}_{P'} \cup \mathcal{ST}_{NLP(P')} = \{\{c,d\}\}$. Hence, $WGCWA^\neg(P) = \{\neg c, \neg d\}$, while $WGCWA^\neg(P') = \{\neg a, \neg b\}$. □

Thus, in the presence of default negation in a program, the monotonic decreasing property of the WGCWA does not hold any more.

For coherent normal logic programs, the three rules coincide.

Proposition 3.13 Let P be a coherent normal logic program. Then, $GCWA^\neg(P) = WGCWA^\neg(P) = PWA^\neg(P)$.

Proof: For a coherent normal logic program P, $\mathcal{ST}_P \cup \mathcal{ST}_{NLP(P)} = \mathcal{ST}_P$. Then the relation $GCWA^\neg(P) = WGCWA^\neg(P)$ holds by each definition. The relation $GCWA^\neg(P) = PWA^\neg(P)$ also holds since $\mathcal{ST}_P = \mathcal{PM}_P$ holds for a coherent normal logic program P. □

The next theorem presents the relationship among the three rules in normal disjunctive programs.

Theorem 3.14 Let P be a coherent normal disjunctive program. Then,

(i) $WGCWA^\neg(P) \subseteq GCWA^\neg(P)$.

(ii) $PWA^\neg(P) \subseteq GCWA^\neg(P)$.

Proof: Since $\mathcal{ST}_P \subseteq \mathcal{ST}_P \cup \mathcal{ST}_{NLP(P)}$, (i) follows from definitions. The part (ii) also follows from the fact that $\mathcal{ST}_P \subseteq \mathcal{PM}_P$. □

As for the WGCWA$^\neg$ and the PWA$^\neg$, there is no inclusion relationship.

Example 3.8 Consider the program:

$$P = \{\ a \vee b \vee c \leftarrow not\, d,\quad e \leftarrow a \wedge b \wedge not\, c\ \}.$$

Then $\mathcal{ST}_P = \{\{a\},\{b\},\{c\}\}$, $\mathcal{ST}_{NLP(P)} = \{\{a,b,c\}\}$, and $WGCWA^\neg(P) = \{\neg d, \neg e\}$, while $\mathcal{PM}_P = \{\{a\},\{b\},\{c\},\{a,b,e\},\{b,c\},\{c,a\},\{a,b,c\}\}$ and $PWA^\neg(P) = \{\neg d\}$. Hence, $WGCWA^\neg(P) \not\subseteq PWA^\neg(P)$. The converse inclusion relation does not hold by Theorem 3.8 either, since each rule reduces to the WGCWA or the PWA in positive disjunctive programs. □

In the above example, $WGCWA^\neg(P)$ treats the disjunction $a \vee b \vee c$ inclusively, then it infers $\neg e$. This is also the case for $GCWA^\neg(P)$ which treats it exclusively. On the other hand, there is the possible model $\{a,b,e\}$ in which a and b are inclusively true and c is exclusively false at the same time, then $\neg e$ is not inferred by $PWA^\neg(P)$. This example illustrates that the possible model semantics also properly treats both types of disjunctions in normal disjunctive programs and provides the most careful negative inference compared with the other two rules.

Moreover, the PWA$^\neg$ can often infer proper negation even in an incoherent program.

Example 3.9 Let P be the program:

$$\{\ a \vee b \leftarrow,\quad b \leftarrow a,\quad \leftarrow not\, a,\quad c \leftarrow not\, b\ \}.$$

Then $\mathcal{ST}_P = \emptyset$, $\mathcal{ST}_{NLP(P)} = \{\{a,b\}\}$, and $\mathcal{PM}_P = \{\{a,b\}\}$, hence $GCWA^\neg(P)$ is not well-defined, while $PWA^\neg(P)$ and $WGCWA^\neg(P)$ imply $\neg c$. □

The above program is incoherent, but p-coherent, since $\{a,b\}$ is a possible model of P which is not a stable model. Observing the above program, the third clause asserts that a should be true, which possibly holds by the first disjunctive clause. Also the truth of a implies the truth of b in the second clause, then it seems natural to assert the falsity of c by the last clause.

As shown in the above example, the disjunctive stable model semantics often fails to capture the intended meaning of a program because of its minimal feature. By contrast, the possible model semantics is well-defined whenever a stable model exists, and is often useful than the disjunctive stable model semantics thanks to its non-minimal nature.

3.4 Fixpoint Semantics

In this section, we present a new fixpoint semantics for disjunctive programs to characterize the possible model semantics presented in the previous sections. We first give a fixpoint semantics for positive disjunctive programs, and generalize it to normal disjunctive programs.

3.4.1 Fixpoint Semantics for Positive Disjunctive Programs

In a definite logic program, van Emden and Kowalski [1976] introduced a fixpoint operator which computes the definite consequences of the program. However, the

situation becomes more complicated in the presence of indefinite consequences in disjunctive programs. In order to characterize such non-deterministic behavior of disjunctive programs, we first introduce a closure operator which operates over the lattice of sets of Herbrand interpretations $2^{2^{\mathcal{HB}_P}}$.

Definition 3.7 Let P be a positive disjunctive program and $\mathcal{I}$ be a set of interpretations. Then a mapping $\mathcal{T}_P : 2^{2^{\mathcal{HB}_P}} \rightarrow 2^{2^{\mathcal{HB}_P}}$ is defined as

$$\mathcal{T}_P(\mathcal{I}) = \bigcup_{I \in \mathcal{I}} T_P(I)$$

where the mapping $T_P : 2^{\mathcal{HB}_P} \rightarrow 2^{2^{\mathcal{HB}_P}}$ is defined as follows:

$$T_P(I) = \begin{cases} \emptyset, & \text{if } \{B_1, \ldots, B_m\} \subseteq I \text{ for some ground integrity constraint} \\ & \quad \leftarrow B_1 \wedge \ldots \wedge B_m \text{ from } P; \\ \{\, J \mid & \text{for each ground clause } C_i : A_1 \vee \ldots \vee A_{l_i} \leftarrow B_1 \wedge \ldots \wedge B_{m_i} \\ & \quad \text{from } P \text{ such that } \{B_1, \ldots, B_{m_i}\} \subseteq I, \\ & \quad J = I \cup \bigcup_{C_i} \{A_j\} \ (1 \leq j \leq l_i) \,\}, & \text{otherwise.} \end{cases} \quad \square$$

Thus, $T_P(I)$ is the set of interpretations J's such that for each clause C_i whose body is satisfied by I, I is expanded into J by adding one disjunct A_j from the heads of every such C_i. In particular, if I does not satisfy an integrity constraint from P, I is removed in $T_P(I)$.

Example 3.10 Let P be the program:

$$\{\, a \vee b \leftarrow c, \quad d \leftarrow c, \quad c \leftarrow, \quad \leftarrow a \wedge b \,\}.$$

Then, $T_P(\{c\}) = \{\{c,d,a\},\{c,d,b\}\}$ and $\mathcal{T}_P(\{\{c,d,a\},\{c,d,b\}\}) = \{\{c,d,a\},\{c,d,b\},\{c,d,a,b\}\}$. $\square$

Definition 3.8 The ordinal powers of $\mathcal{T}_P$ are defined as follows:

$$\begin{array}{lcl} \mathcal{T}_P \uparrow 0 & = & \{\emptyset\}, \\ \mathcal{T}_P \uparrow n+1 & = & \mathcal{T}_P(\mathcal{T}_P \uparrow n), \\ \mathcal{T}_P \uparrow \omega & = & \bigcup_{\alpha<\omega} \bigcap_{\alpha \leq n < \omega} \mathcal{T}_P \uparrow n, \end{array}$$

where n is a successor ordinal and ω is a limit ordinal. $\square$

The above definition means that at the limit ordinal ω the closure retains interpretations which are persistent in the preceding iterations. That is, for any interpretation I in $\mathcal{T}_P \uparrow \omega$, there is an ordinal α smaller than ω such that for every n $(\alpha \leq n < \omega)$, I is included in $\mathcal{T}_P \uparrow n$. Such a closure definition is also used in [Fages, 1991; Teusink, 1993a] for computing stable models of normal logic programs.

Theorem 3.15 $\mathcal{T}_P \uparrow \omega$ is a fixpoint.

Proof: When $I \in \mathcal{T}_P \uparrow \omega$, suppose that there is no interpretation J in $\mathcal{T}_P \uparrow \omega$ such that $I \in \mathcal{T}_P(\{J\})$. In this case, for any α there is some n $(\alpha \leq n < \omega)$ such that J is not included in $\mathcal{T}_P \uparrow n$. Then $I \notin \mathcal{T}_P \uparrow n+1$. This contradicts the fact that $I \in \mathcal{T}_P \uparrow \omega$. Thus, $J \in \mathcal{T}_P \uparrow \omega$, so $I \in \mathcal{T}_P(\mathcal{T}_P \uparrow \omega)$. Conversely, if $I \in \mathcal{T}_P(\mathcal{T}_P \uparrow \omega)$, there is an interpretation J in $\mathcal{T}_P \uparrow \omega$ such that $I \in \mathcal{T}_P(\{J\})$. Then J is included in any $\mathcal{T}_P \uparrow n$ for $\alpha \leq n < \omega$ by definition. Thus $I \in \mathcal{T}_P \uparrow n$ for any $\alpha + 1 \leq n < \omega$. Hence, $I \in \mathcal{T}_P \uparrow \omega$. $\square$

Example 3.11 (cont. from Example 3.10) Given the program P, it becomes

$$\begin{array}{l} \mathcal{T}_P \uparrow 1 = \{\{c\}\}, \\ \mathcal{T}_P \uparrow 2 = \{\{c,d,a\},\{c,d,b\}\}, \\ \mathcal{T}_P \uparrow 3 = \{\{c,d,a\},\{c,d,b\},\{c,d,a,b\}\}, \\ \mathcal{T}_P \uparrow 4 = \{\{c,d,a\},\{c,d,b\},\{c,d,a,b\}\}, \end{array}$$

where $\mathcal{T}_P \uparrow \omega = \mathcal{T}_P \uparrow 3$. $\square$

In the above example, the interpretation $\{c,d,a,b\}$ in $\mathcal{T}_P \uparrow 3$ is pruned in $\mathcal{T}_P \uparrow 4$ by the integrity constraint $\leftarrow a \wedge b$, while the same interpretation is also generated from $\{c,d,a\}$ and $\{c,d,b\}$ in $\mathcal{T}_P \uparrow 3$, hence $\{c,d,a,b\}$ remains in $\mathcal{T}_P \uparrow 4$.

By definition, the fixpoint closure presented above exists for any program and is uniquely determined. Intuitively, the fixpoint characterizes a set of interpretations which are "generated" in a program by starting from the empty interpretation. Next we show that the fixpoint closure in fact contains what we want, i.e., the set of all possible models.

Lemma 3.16 Let P be a positive disjunctive program. Then I is a model of P iff $I \in \mathcal{T}_P(\{I\})$.

Proof: I is a model of P
iff it satisfies integrity constraints and for each clause $A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m$ in P, $\{B_1, \ldots, B_m\} \subseteq I$ implies $A_i \in I$ for some A_i $(1 \leq i \leq l)$
iff $I \in \mathcal{T}_P(\{I\})$. $\square$

Let $\mu(\mathcal{T}_P \uparrow \omega) = \{\, I \mid I \in \mathcal{T}_P \uparrow \omega \text{ and } I \in \mathcal{T}_P(\{I\}) \,\}$. Then, by Lemma 3.16, $\mu(\mathcal{T}_P \uparrow \omega)$ represents the set of models of P which are included in the fixpoint closure. Also let $min(\mathcal{I}) = \{\, I \in \mathcal{I} \mid \nexists J \in \mathcal{I} \text{ such that } J \subset I \,\}$. Then the following relations hold.

Theorem 3.17 Let P be a positive disjunctive program. Then,

(i) $\mathcal{PM}_P = \mu(\mathcal{T}_P \uparrow \omega)$.

(ii) $\mathcal{MM}_P = min(\mu(\mathcal{T}_P \uparrow \omega))$.

Proof: (i) I is in $\mu(\mathcal{T}_P \uparrow \omega)$
iff I is included in $\mathcal{T}_P \uparrow \omega$ and is a model of P (by Lemma 3.16)
iff each A_i in I is included in the derived head of a ground clause $A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m$ $(1 \leq i \leq l)$ in P and I satisfies every integrity constraint in P
iff I is the least Herbrand model of a consistent split Horn logic program P_s of P and each A_i in I is derived by the split clause $A_i \leftarrow B_1 \wedge \ldots \wedge B_m$ in P_s
iff I is in $\mathcal{PM}_P$.

(ii) The result follows from (i) and Proposition 3.5. □

Corollary 3.18 A positive disjunctive program P is inconsistent iff $\mu(\mathcal{T}_P \uparrow \omega) = \emptyset$. □

Example 3.12 (cont. from Example 3.11)
$\mu(\mathcal{T}_P \uparrow \omega) = \{\{c, d, a\}, \{c, d, b\}\}$, which coincides with the set of all possible models of P. □

For definite logic programs, our fixpoint construction reduces to van Emden and Kowalski's fixpoint semantics [van Emden and Kowalski, 1976].

Corollary 3.19 Let P be a definite logic program. Then $\mathcal{T}_P \uparrow \omega$ contains the unique least Herbrand model of P. □

3.4.2 Fixpoint Semantics for Normal Disjunctive Programs

Now we extend the fixpoint semantics of positive disjunctive programs to normal disjunctive programs. To this end, we first introduce a program transformation which translates a normal disjunctive program into a semantically equivalent positive disjunctive program.[2]

[2]The transformation is originally introduced in [Inoue et al., 1992] in a different form.

Definition 3.9 Let P be a normal disjunctive program. Then its *epistemic transformation* is defined as the positive disjunctive program P^κ obtained from P by replacing each clause containing default negation:

$$A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\ B_{m+1} \wedge \ldots \wedge not\ B_n \quad (m \neq n) \tag{3.1}$$

with the following *not*-free clauses in P^κ:

$$\lambda_1 \vee \ldots \vee \lambda_l \vee \mathsf{K}B_{m+1} \vee \ldots \vee \mathsf{K}B_n \leftarrow B_1 \wedge \ldots \wedge B_m\,, \tag{3.2}$$

$$A_i \leftarrow \lambda_i \quad \text{for } i = 1, \ldots, l\,, \tag{3.3}$$

$$\leftarrow \lambda_i \wedge B_j \quad \text{for } i = 1, \ldots, l \text{ and } j = m+1, \ldots, n\,, \tag{3.4}$$

$$\lambda_i \leftarrow A_i \wedge \lambda_k \quad \text{for } i = 1, \ldots, l \text{ and } k = 1, \ldots, l\,. \tag{3.5}$$

In particular, each integrity constraint containing default negation is transformed into

$$\mathsf{K}B_{m+1} \vee \ldots \vee \mathsf{K}B_n \leftarrow B_1 \wedge \ldots \wedge B_m\,.$$

Note here that each *not*-free clause in P is included in P^κ as it is. □

In the epistemic transformation, the newly introduced atom $\mathsf{K}B_j$ means that B_j *is believed*. With this epistemic reading, each default negation *not* B_j in the body of a clause is rewritten in $\neg\mathsf{K}B_j$ which means that B_j *is disbelieved*,[3] and shifted to the head of the clause. Also each λ_i is a newly introduced atom appearing nowhere in P and is uniquely associated with each ground instance of a clause (3.1) from P.[4]

An intuitive reading of the transformed clauses is that if $B_1, \ldots, B_m$ are true, then some A_i $(1 \leq i \leq l)$ becomes true via λ_i when $B_{m+1}, \ldots, B_n$ are not true; otherwise, some B_j $(m+1 \leq j \leq n)$ is believed. The clause (3.5) has an effect to associate λ_i with A_i whenever A_i is true and another disjunct A_k is derived from (3.2) via λ_k.[5] In this way, every normal disjunctive program P is transformed into a positive disjunctive program P^κ. Then we can construct the fixpoint of P^κ as presented in the previous section.

Let I^κ be an interpretation of P^κ. Then I^κ is called *canonical* if $\mathsf{K}A \in I^\kappa$ implies $A \in I^\kappa$ for any atom A in $\mathcal{HB}_P$. That is, in a canonical interpretation each believed atom has a justification. Given a set of interpretations $\mathcal{I}_{P^\kappa}$, let

$$obj_c(\mathcal{I}_{P^\kappa}) = \{I^\kappa \cap \mathcal{HB}_P \mid I^\kappa \in \mathcal{I}_{P^\kappa} \text{ and } I^\kappa \text{ is canonical }\}.$$

[3]Such an interpretation of default negation is firstly proposed by [Gelfond, 1987].

[4]If a clause contains n distinct free variables $\mathbf{x} = x_1, \ldots, x_n$, then a new atom $\lambda_i(\mathbf{x})$ can be associated with each A_i, where λ_i is an n-ary predicate symbol appearing nowhere in P.

[5]In case of $l = 1$, the clause (3.5) becomes a tautological clause $\lambda \leftarrow A \wedge \lambda$ and hereafter we will omit such a clause in P^κ.

Now we present the fixpoint characterization of possible models in normal disjunctive programs. We first prove preliminary lemmas.

Lemma 3.20 Let P be a normal disjunctive program. Then,

$$\mathcal{ST}_P = obj_c(min(\mu(\mathcal{T}_{P^\kappa} \uparrow \omega))).$$

Proof: Suppose that I is in $obj_c(min(\mu(\mathcal{T}_{P^\kappa} \uparrow \omega)))$. Let I^κ be a canonical interpretation in $min(\mu(\mathcal{T}_{P^\kappa} \uparrow \omega))$ such that $I^\kappa \cap \mathcal{HB}_P = I$. Then, for each ground clause of the form (3.1) from P, $\{B_1, \ldots, B_m\} \subseteq I^\kappa$ implies either (i) $\exists \lambda_i \in I^\kappa$ $(1 \leq i \leq l)$, $A_i \in I^\kappa$, and $\{B_{m+1}, \ldots, B_n\} \cap I^\kappa = \emptyset$, or (ii) $\exists \mathsf{K}B_j \in I^\kappa$ $(m+1 \leq j \leq n)$ by (3.2), (3.3), and (3.4).[6]

In case of (i), $\{B_{m+1}, \ldots, B_n\} \cap I^\kappa = \emptyset$ implies $\{B_{m+1}, \ldots, B_n\} \cap I = \emptyset$. Then there is a clause of the form:

$$A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m \quad (*)$$

in P^I. Since $\{B_1, \ldots, B_m\} \subseteq I^\kappa$ and $A_i \in I^\kappa$ implies $\{B_1, \ldots, B_m\} \subseteq I$ and $A_i \in I$, I satisfies the clause $(*)$ in P^I. In case of (ii), since I^κ is canonical, $\mathsf{K}B_j \in I^\kappa$ implies $B_j \in I^\kappa$, and thus $B_j \in I$. In this case, the clause $(*)$ is not included in P^I. Thus, in both cases, I satisfies every clause in P^I.

Suppose that there is an interpretation J such that (a) $J \subset I$ and (b) J satisfies each clause from P^I. Then, two conditions (a) and (b) are satisfied only if there is a clause $(*)$ such that $\{B_1, \ldots, B_m\} \subseteq J$ and for some two atoms A_{i_1} and A_{i_2} $(1 \leq i_1, i_2 \leq l; i_1 \neq i_2)$, $A_{i_1} \in J$ but $A_{i_2} \in I \setminus J$. Without loss of generality, we can assume that just one such clause exists in P^I. Since I does not contain atoms $B_{m+1}, \ldots, B_n$, the corresponding canonical interpretation I^κ does not contain $\mathsf{K}B_{m+1}, \ldots, \mathsf{K}B_n$ either. Thus, $\{B_1, \ldots, B_m\} \subseteq I$ implies $\exists \lambda_k \in I^\kappa$ for some $1 \leq k \leq l$. Since $A_{i_1}, A_{i_2} \in I$ implies $A_{i_1}, A_{i_2} \in I^\kappa$, $\lambda_k \in I^\kappa$ implies $\lambda_{i_1}, \lambda_{i_2} \in I^\kappa$ by (3.5). Let $J^\kappa = I^\kappa \setminus \{A_{i_2}, \lambda_{i_2}\}$. Then, the interpretation J^κ satisfies all the clauses (3.2), (3.3), (3.4), (3.5) in P^κ. This contradicts the fact that I^κ is a minimal model of P^κ. Then I is also a minimal model of P^I, hence a stable model of P.

[6] When a clause (3.1) contains no *not*, $\{B_1, \ldots, B_m\} \subseteq I^\kappa$ implies $A_i \in I^\kappa$ $(1 \leq i \leq l)$ as a special case of (i).

Conversely, suppose that I is a stable model of P. Then, for each clause C of the form (3.1) from P, let $I_\lambda = \bigcup_C \{\lambda_i \mid \{B_1, \ldots, B_m\} \subseteq I, \{B_{m+1}, \ldots, B_n\} \cap I = \emptyset$, and $A_i \in I$ $(1 \leq i \leq l)\}$ and $I_K = \bigcup_C \{\mathsf{K}B_j \mid \{B_1, \ldots, B_m\} \subseteq I$ and $B_j \in I$ $(m+1 \leq j \leq n)\}$. Let $I^{\kappa\prime} = I \cup I_\lambda \cup I_K$. Then, $I^{\kappa\prime}$ satisfies each clause (3.2), (3.3), (3.4), and (3.5) from P^κ, and by the construction of $I^{\kappa\prime}$, $I^{\kappa\prime} \in \mu(\mathcal{T}_{P^\kappa} \uparrow \omega)$. Now let us define $I^\kappa = I \cup S$ where S is a minimal subset of $I_\lambda \cup I_K$ such that each λ_i or $\mathsf{K}B_j$ is chosen in a way that I^κ satisfies every clause in P^κ. Note that for each atom $\mathsf{K}B$ in I^κ, $B \in I$ by definition, so $B \in I^\kappa$. Hence I^κ is canonical. Next assume that there exists $J^\kappa \in \mu(\mathcal{T}_{P^\kappa} \uparrow \omega)$ such that $J^\kappa \subset I^\kappa$. Since we have defined I^κ as a minimal set with respect to the atoms from $I_\lambda \cup I_K$, the inclusion relation implies $J^\kappa \cap \mathcal{HB}_P \subset I^\kappa \cap \mathcal{HB}_P$. Then $\exists A_i \in I^\kappa \setminus J^\kappa$. In this case, there is a clause (3.2) in P^κ such that $\{B_1, \ldots, B_m\} \subseteq J^\kappa$, $\lambda_i \in I^\kappa \setminus J^\kappa$, $\mathsf{K}B_j \in J^\kappa$ for some $1 \leq i \leq l$ and $m+1 \leq j \leq n$. Since $J^\kappa \subset I^\kappa$, $\mathsf{K}B_j \in I^\kappa$. As I^κ is canonical, $\mathsf{K}B_j \in I^\kappa$ implies $B_j \in I^\kappa$. But this is impossible from the condition (3.4). Thus, there is no J^κ which is smaller than I^κ, hence $I^\kappa \in min(\mu(\mathcal{T}_{P^\kappa} \uparrow \omega))$. Since I^κ is canonical, $I \in obj_c(min(\mu(\mathcal{T}_{P^\kappa} \uparrow \omega)))$. □

Lemma 3.21 Let P be a normal logic program. Then,

$$\mathcal{ST}_P = obj_c(\mu(\mathcal{T}_{P^\kappa} \uparrow \omega)).$$

Proof: By Lemma 3.20, $obj_c(min(\mu(\mathcal{T}_{P^\kappa} \uparrow \omega)))$ is the set of all stable models of P. Since $I \in obj_c(min(\mu(\mathcal{T}_{P^\kappa} \uparrow \omega)))$ implies $I \in obj_c(\mu(\mathcal{T}_{P^\kappa} \uparrow \omega))$, we show that the converse is also true. Assume that the converse does not hold. That is, there is a non-minimal set $I \in obj_c(\mu(\mathcal{T}_{P^\kappa} \uparrow \omega))$ and $\exists J \in obj_c(min(\mu(\mathcal{T}_{P^\kappa} \uparrow \omega)))$ such that $J \subset I$. In this case, there exists an atom A such that $A \in I \setminus J$. Let $I = I^\kappa \cap \mathcal{HB}_P$ and $J = J^\kappa \cap \mathcal{HB}_P$ for some canonical interpretations I^κ and J^κ. Then, corresponding to (3.2), (3.3), and (3.4), there exist clauses:

$$\lambda \vee \mathsf{K}B_{m+1} \vee \ldots \vee \mathsf{K}B_n \leftarrow B_1 \wedge \ldots \wedge B_m,$$

$$A \leftarrow \lambda,$$

$$\leftarrow \lambda \wedge B_j \quad (j = m+1, \ldots, n)$$

in P^κ, where $\{B_1, \ldots, B_m\} \subseteq I^\kappa$, $\{B_1, \ldots, B_m\} \subseteq J^\kappa$, $\lambda \in I^\kappa$, and $\exists \mathsf{K}B_j \in J^\kappa$ $(m+1 \leq j \leq n)$. Note here that the clause (3.5) becomes $\lambda \leftarrow A \wedge \lambda$ and is neglected. Since J^κ is canonical, $B_j \in J^\kappa$. Then $J \subset I$ implies $B_j \in I^\kappa$. But this is impossible from the third clause above. □

Now we are ready to prove the main theorem.

Theorem 3.22 Let P be a normal disjunctive program. Then,

$$\mathcal{PM}_P = obj_c(\mu(\mathcal{T}_{P^\kappa} \uparrow \omega)).$$

Proof: Let I be a possible model of P. Then I is a stable model of some coherent split normal logic program P_s of P. By Lemma 3.21, for the transformed program P_s^κ of P_s, I is in $obj_c(\mu(\mathcal{T}_{P_s^\kappa} \uparrow \omega))$. Since P_s is a program obtained by splitting each disjunctive clause in P, every interpretation included in $\mathcal{T}_{P_s^\kappa} \uparrow \omega$ is also obtained by splitting during the computation of $\mathcal{T}_{P^\kappa} \uparrow \omega$. Moreover, since I satisfies each integrity constraint in P_s^κ, it also satisfies the same integrity constraints in P^κ. Hence I is also in $obj_c(\mu(\mathcal{T}_{P^\kappa} \uparrow \omega))$. The converse is also shown in the same manner. □

The above lemmas and theorem are direct extensions of Theorem 3.17.

Corollary 3.23 Let P be a normal disjunctive program. Then,

(i) P is inconsistent iff $\mu(\mathcal{T}_{P^\kappa} \uparrow \omega) = \emptyset$.

(ii) P is incoherent iff $obj_c(min(\mu(\mathcal{T}_{P^\kappa} \uparrow \omega))) = \emptyset$.

(iii) P is p-incoherent iff $obj_c(\mu(\mathcal{T}_{P^\kappa} \uparrow \omega)) = \emptyset$. □

Example 3.13 (cont. from Example 3.4) The program

$$P = \{\ a \vee b \leftarrow,\quad b \leftarrow a,\quad c \leftarrow not\ a\ \}$$

is transformed into the epistemic form:

$$P^\kappa = \{\ a \vee b \leftarrow,\quad b \leftarrow a,\quad \lambda \vee \mathsf{K}a \leftarrow,\quad c \leftarrow \lambda,\quad \leftarrow \lambda \wedge a\ \}.$$

Then it becomes

$$\mu(\mathcal{T}_{P^\kappa} \uparrow \omega) = \{\{a, b, \mathsf{K}a\}, \{b, \mathsf{K}a\}, \{b, c, \lambda\}, \{b, c, \lambda, \mathsf{K}a\}\}.$$

Thus,

$$obj_c(\mu(\mathcal{T}_{P^\kappa} \uparrow \omega)) = \{\{a, b\}, \{b, c\}\},$$

which contains the possible models of P. On the other hand,

$$min(\mu(\mathcal{T}_{P^\kappa} \uparrow \omega)) = \{\{b, \mathsf{K}a\}, \{b, c, \lambda\}\},$$

then

$$obj_c(min(\mu(\mathcal{T}_{P^\kappa} \uparrow \omega))) = \{\{b, c\}\},$$

which contains the stable model of P. □

As noticed in the previous section, stable models are not necessarily minimal possible models. However, the above example shows that we can compute stable models exactly by computing the minimal set of the closure $\mu(\mathcal{T}_{P^\kappa} \uparrow \omega)$ before applying the operation $obj_c(\cdot)$.

3.5 Computing Possible Models

In this section, we provide a bottom-up proof procedure to compute possible models of disjunctive programs. We assume in this section that a program is function-free and range-restricted, that is, a program containing no function symbol and any variable in a clause has an occurrence in a positive atom in the body. Such conditions are usually imposed on a program in the context of deductive databases.

3.5.1 Bottom-up Model Generation Procedure

The algorithm we use to compute possible models in disjunctive programs is based on a *bottom-up model generation proof procedure*.

Let P be a positive disjunctive program and $\mathcal{I}_P^i$ be a set of interpretations of P. Let $\mathcal{I}_P^0 = \{\emptyset\}$. For $i \geq 0$ do:

1. For any $I \in \mathcal{I}_P^i$, if there is an integrity constraint in P of the form:

   $$\leftarrow B_1 \wedge \ldots \wedge B_m$$

   such that $I \models (B_1 \wedge \ldots \wedge B_m)\sigma$ for some ground substitution σ, then remove I from $\mathcal{I}_P^i$.

2. For any $I \in \mathcal{I}_P^i$, for every clause C_k in P of the form:

   $$C_k:\ A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m\ \ (l \geq 1)$$

   such that $I \models (B_1 \wedge \ldots \wedge B_m)\sigma$ for some ground substitution σ, put $I \cup \bigcup_{C_k}\{A_j\sigma\}$ into $\mathcal{I}_P^{i+1}$ for every $j = 1, \ldots, l$.

3. Iterate the above two steps until it reaches the fixpoint $\mathcal{I}_P^{n+1} = \mathcal{I}_P^n$ which is closed under the above two operations.

In step 1, the procedure prunes interpretations which do not satisfy integrity constraints in the program. In step 2, the procedure generates the new set of interpretations $\mathcal{I}_P^{i+1}$ from the given interpretations $\mathcal{I}_P^i$ by performing forward reasoning

based on hyperresolution[7] and case-splitting on non-unit derived clauses. Note here that since a program is range-restricted, each disjunct $A_j\sigma$ generated in step 2 is always ground. Hence the soundness for unsatisfiability by case-splitting is guaranteed [Manthey and Bry, 1988]. Moreover, since we consider a finite function-free program, the above procedure always terminates in a finite step.

The connection between the above closure computation and the fixpoint semantics with the mapping $\mathcal{T}_P$ given in Section 3.4.1 is obvious. This correspondence can also be regarded as an extension of the relation between hyperresolution and van Emden and Kowalski's fixpoint semantics for definite logic programs [van Emden and Kowalski, 1976, Section 8].

Now we characterize the possible model semantics using the algorithm presented above. Let $\mathcal{I}_P^\omega$ be the fixpoint closure obtained by the above procedure. Then the following results directly follow from Theorem 3.17.

Theorem 3.24 Let P be a positive disjunctive program. Then the following relations hold.

(i) $\mathcal{PM}_P = \mu(\mathcal{I}_P^\omega)$.

(ii) $\mathcal{MM}_P = min(\mu(\mathcal{I}_P^\omega))$.

In particular, P is inconsistent iff $\mu(\mathcal{I}_P^\omega) = \emptyset$. □

Corollary 3.25 For a consistent positive disjunctive program P and a ground atom A,

(i) $GCWA(P) \models \neg A$ iff $A \notin I$ for any $I \in min(\mu(\mathcal{I}_P^\omega))$.

(ii) $WGCWA(P) \models \neg A$ iff $A \notin I$ for $I \in \mathcal{I}_{Horn(P)}^\omega$.

(iii) $PWA(P) \models \neg A$ iff $A \notin I$ for any $I \in \mu(\mathcal{I}_P^\omega)$.

Proof: (i) and (iii) directly follow from each definition and the results in Theorem 3.24. Since $\mathcal{I}_{Horn(P)}^\omega$ contains a unique element which is the least Herbrand model of $Horn(P)$, (ii) also follows from the definition of the WGCWA. □

For normal disjunctive programs, the following results hold by Lemmas 3.20, 3.21 and Theorem 3.22.

[7] $A_1 \vee \ldots \vee A_l$ is said to be obtained from $A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m$ and $B_1, \ldots, B_m$ by *hyperresolution* [Robinson, 1965b].

Theorem 3.26 Let P be a normal disjunctive program and P^κ be its epistemic transformation. Then the following relations hold.

(i) $\mathcal{PM}_P = obj_c(\mu(\mathcal{I}_{P^\kappa}^\omega))$.

(ii) $\mathcal{ST}_P = obj_c(min(\mu(\mathcal{I}_{P^\kappa}^\omega)))$.

(iii) For a normal logic program P, $\mathcal{PM}_P = \mathcal{ST}_P = obj_c(\mu(\mathcal{I}_{P^\kappa}^\omega))$.

In particular, P is p-incoherent (resp. incoherent) iff $obj_c(\mu(\mathcal{I}_{P^\kappa}^\omega)) = \emptyset$ (resp. $obj_c(min(\mu(\mathcal{I}_{P^\kappa}^\omega))) = \emptyset$). □

Corollary 3.27 Let P be a normal disjunctive program and A be a ground atom.

(i) For a coherent program P, $GCWA^\neg(P) \models \neg A$ iff $A \notin I$ for any $I \in obj_c(min(\mu(\mathcal{I}_{P^\kappa}^\omega)))$.

(ii) For a weakly coherent program P, $WGCWA^\neg(P) \models \neg A$ iff $A \notin I$ for any $I \in obj_c(min(\mu(\mathcal{I}_{P^\kappa}^\omega))) \cup obj_c(\mu(\mathcal{I}_{NLP(P)^\kappa}^\omega))$.

(iii) For a p-coherent program P, $PWA^\neg(P) \models \neg A$ iff $A \notin I$ for any $I \in obj_c(\mu(\mathcal{I}_{P^\kappa}^\omega))$.

Proof: (i) and (iii) directly follow from Theorem 3.26 (i) and (ii). (ii) also follows from Theorem 3.26 (iii) and the definition of the WGCWA$^\neg$. □

3.5.2 Query Answering

In this section, we address an application of the previously presented algorithm to query answering under the possible model semantics in normal disjunctive programs.

A *query* we consider here is the following form:

$$Q(\mathbf{x}) \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\ B_{m+1} \wedge \ldots \wedge not\ B_n \tag{3.6}$$

where (3.6) is a function-free range-restricted normal clause and $\mathbf{x}$ represents variables appearing in the body of the clause. An *answer* to the query is a ground substitution σ for variables in $Q(\mathbf{x})$. In particular, if Q contains no variable, σ is the empty substitution.

For a given normal disjunctive program P, let P_Q be a program obtained from P by adding a query of the form (3.6). Then, the query is *true* in P under the possible model semantics if for every possible model I of P_Q there is an answer σ such that $Q(\mathbf{x})\sigma$ is included in I. Else if for some possible model I of P_Q there is an answer σ such that $Q(\mathbf{x})\sigma$ is included in I, a query is *possibly true*. Otherwise, if there is no such answer, a query is *false*. By Theorem 3.26 (i), the following results hold.

Theorem 3.28 Let P be a normal disjunctive program and Q be a query. Then,

(i) Q is true in P iff for any $I \in obj_c(\mu(\mathcal{I}^{\omega}_{P_Q{}^{\kappa}}))$, $Q(\mathbf{x})\sigma \in I$ for some σ.

(ii) Q is possibly true in P iff for some $I \in obj_c(\mu(\mathcal{I}^{\omega}_{P_Q{}^{\kappa}}))$, $Q(\mathbf{x})\sigma \in I$ for some σ.

(iii) Q is false in P iff for any $I \in obj_c(\mu(\mathcal{I}^{\omega}_{P_Q{}^{\kappa}}))$, $Q(\mathbf{x})\sigma \notin I$ for any σ. □

Example 3.14 Let P be the program:

$$\{\ p(a) \vee p(b) \leftarrow\ \}.$$

Then, $q_1(x) \leftarrow p(x)$ is true, $q_2 \leftarrow p(a)$ is possibly true, and $q_3 \leftarrow p(c)$ is false. □

By using Theorem 3.26 (ii) instead, the above result is also applicable to query answering under the disjunctive stable model semantics.

3.6 Computational Complexity

In a propositional positive disjunctive program, a minimal model exists whenever the program is satisfiable. Then the complexity of the existence problem (resp. the set-entailment problem) for the minimal model semantics is NP-complete (resp. co-NP-complete). Since the possible model semantics coincides with the minimal model semantics for positive inference, those complexity results also hold for the possible model semantics in propositional positive disjunctive programs. On the other hand, it is known that the complexity of the set-membership problem for the minimal model semantics is Σ^p_2-complete, and thus inferring negation under the GCWA is Π^p_2-complete [Eiter and Gottlob, 1993c]. By contrast, Chan [1993] has shown that, in a propositional positive disjunctive program, inferring negation under the WGCWA or the PWA is still co-NP-complete. In particular, in the absence of integrity constraints both the WGCWA and the PWA are tractable.

These observations tell us that the possible model semantics has the computational advantage over the minimal model semantics for inferring negation, since it does not increase the complexity more than the classical propositional entailment. Moreover, as shown in Section 3.2, since the PWA is more intuitive than the WGCWA, it is concluded that the possible model semantics is the best choice among others from both the reasoning and computational points of view.

In this section, we prove that the complexity results for the possible model semantics is still within (co-)NP, even in normal disjunctive programs. We show this fact by transforming possible models in a normal disjunctive program into stable models in a normal logic program.

Definition 3.10 Let P be a normal disjunctive program. Then the *pm-transformation* transforms P into the normal logic program $\wp(P)$ which is obtained from P by replacing each disjunctive clause:

$$A_1 \vee \ldots \vee A_l \leftarrow \Gamma \tag{3.7}$$

in P with the following normal clauses and an integrity constraint:

$$A_i \leftarrow \Gamma \wedge not\ A'_i \quad \text{for } i = 1, \ldots, l, \tag{3.8}$$
$$A'_i \leftarrow \Gamma \wedge not\ A_i \quad \text{for } i = 1, \ldots, l, \tag{3.9}$$
$$\leftarrow \Gamma \wedge A'_1 \wedge \ldots \wedge A'_l, \tag{3.10}$$

in $\wp(P)$ where Γ denotes a conjunction in the body of the clause and each A'_i is a new atom not appearing in P and is uniquely introduced for each A_i in $\mathcal{HB}_P$. □

The intuitive meaning of the *pm*-transformation is that: when A_i becomes true by the disjunctive clause (3.7), we can make it true also by the corresponding normal clause (3.8) in $\wp(P)$ by assuming that its complementary atom A'_i is not true. Else when A_i does not become true by (3.7), we will make A'_i true by assuming that A_i is not true in the corresponding normal clause (3.9) in $\wp(P)$. The condition (3.10) states that when Γ is true, every A'_i cannot become true at the same time, that is, at least one A_i should be true. Thus, the transformed clauses represent every possible selection of disjuncts from the disjunctive head of the clause, which exactly characterizes every set of split clauses of (3.7).

Now we show that there is a one-to-one correspondence between the possible models of P and the stable models of $\wp(P)$. We first present a preliminary lemma.

Lemma 3.29 Let P be a normal disjunctive program. Then M is a possible model of P iff M is a possible model of P^M.

Proof: M is a possible model of P
iff M is a stable model of some split normal logic program P_s of P
iff M is the least Herbrand model of $P_s{}^M$
iff M is the least Herbrand model of some split Horn logic program P^M_s of P^M
iff M is a possible model of P^M. □

Theorem 3.30 Let P be a normal disjunctive program and $\wp(P)$ be its *pm*-transformation. Then $\mathcal{PM}_P = \mathcal{ST}_{\wp(P)} \cap \mathcal{HB}_P$ holds, where $\mathcal{ST}_{\wp(P)} \cap \mathcal{HB}_P = \{I \cap \mathcal{HB}_P \mid I \in \mathcal{ST}_{\wp(P)}\}$.

Proof: (i) First we show that $\mathcal{PM}_P = \mathcal{ST}_{\wp(P)} \cap \mathcal{HB}_P$ holds for a positive disjunctive program P. Let M be a possible model of a positive disjunctive program P. Then M is the least Herbrand model of a split program P_s of P. In this case, M is also the least Herbrand model of a program in which each disjunctive clause (3.7) in P is replaced with its split clauses $\{A_i \leftarrow \Gamma \mid A_i \in M \cap \{A_1, \ldots, A_l\}\}$. Now let us consider a Horn logic program P'_s which is obtained from P by replacing each disjunctive clause (3.7) with clauses of $\{A_i \leftarrow \Gamma \mid A_i \in M \cap \{A_1, \ldots, A_l\}\} \cup \{A'_j \leftarrow \Gamma \mid A_j \in \{A_1, \ldots, A_l\} \setminus M\}$. Let M' be the least Herbrand model of P'_s. Then clearly $M = M' \cap \mathcal{HB}_P$ holds. Here a program P'_s together with the integrity constraint (3.10) coincides with the program $\wp(P)^{M'}$. Since $M \models \Gamma$ implies at least one $A_i \in M$, M' satisfies the condition (3.10). Then M' is also the least Herbrand model of $\wp(P)^{M'}$, hence a stable model of $\wp(P)$.

Conversely, let M be a stable model of $\wp(P)$. Since M satisfies the condition (3.10), $M \models \Gamma$ implies that at least one of the clauses (3.8) becomes $A_i \leftarrow \Gamma$ in $\wp(P)^M$, and for each such clause $M \models \Gamma$ implies $A_i \in M$. In this case, there is a split program P_s of P in which each disjunctive clause (3.7) is replaced with its split clauses $\{A_i \leftarrow \Gamma \mid A_i \in M \cap \{A_1, \ldots, A_l\}\}$. Since M is the least Herbrand model of $\wp(P)^M$, $M \cap \mathcal{HB}_P$ is also the least Herbrand model of P_s, hence a possible model of P.

(ii) Next we show that $\mathcal{PM}_P = \mathcal{ST}_{\wp(P)} \cap \mathcal{HB}_P$ holds for a normal disjunctive program P. Let M be a possible model of a normal disjunctive program P. By Lemma 3.29, M is also a possible model of a positive disjunctive program P^M. Then, by (i), there is a stable model M' of $\wp(P^M)$ such that $M = M' \cap \mathcal{HB}_P$, which is also the least Herbrand model of $\wp(P^M)^{M'}$. Since $\wp(P^M)^{M'} = \wp(P^{M'})^{M'} = \wp(P)^{M'}$, M' is also a stable model of $\wp(P)$.

Conversely, let M be a stable model of $\wp(P)$. Then M is the least Herbrand model of $\wp(P)^M$. Since $\wp(P)^M = \wp(P^M)^M$, M is also the least Herbrand model of $\wp(P^M)^M$, and a stable model of $\wp(P^M)$. Then, by (i), $M \cap \mathcal{HB}_P$ is a possible model of P^M. Since $P^M = P^{M \cap \mathcal{HB}_P}$, $M \cap \mathcal{HB}_P$ is a possible model of P by Lemma 3.29. □

The above theorem presents that the possible models of any normal disjunctive program are expressed by the stable models of the corresponding transformed normal logic program.

For the stable model semantics in propositional normal logic programs, both the existence problem and the set-membership problem are NP-complete, while the set-entailment problem is co-NP-complete [Marek and Truszczynski, 1991a; 1991b].[8] We use this fact to show the computational complexity of the possible model semantics.

Theorem 3.31 Let P be a propositional normal disjunctive program. Then,

(i) Deciding the existence of a possible model of P is NP-complete.

(ii) Deciding whether an atom is true in some possible model of P is NP-complete.

(iii) Deciding whether an atom is true in every possible model of P is co-NP-complete.

Proof: Since possible models coincide with stable models in normal logic programs, each decision problem under the possible models semantics is (co-)NP-hard. To see that it is in (co-)NP, the *pm*-transformation efficiently translates each decision problem for possible models into the corresponding problem for stable models which is in (co-)NP, then the membership in (co-)NP follows. □

Corollary 3.32 Inferring negation under the PWA$^\neg$ is co-NP-complete. □

It is known that the decision problems for the disjunctive stable model semantics is Σ_2^P-complete for the existence and the set-membership problem, and Π_2^P-complete for the set-entailment problem [Eiter and Gottlob, 1993a]. Then the following result also follows from the definition.

Corollary 3.33 Inferring negation under the GCWA$^\neg$ or the WGCWA$^\neg$ is both Π_2^P-complete. □

The complexity results for disjunctive programs are summarized in Table 3.1.

These results show that the frameworks based on the minimal/disjunctive stable model semantics introduce an additional source of complexity for minimality-checking, while this is not the case for computation of possible models due to its non-minimal feature.

We have already seen in the previous sections that the possible model semantics can provide flexible reasoning mechanisms compared with the minimal/disjunctive stable model semantics thanks to its non-minimal nature. The results of this section present that this unique feature of the possible model semantics also brings a computational advantage over those minimal model based semantics.

[8] In [Marek and Truszczynski, 1991a; 1991b], integrity constraints are not included in a program. However, a program containing integrity constraints is easily reducible to the one without them by rewriting each integrity constraint $\leftarrow G$ by the normal clause $false \leftarrow G$.

Table 3.1: Complexity Results for Disjunctive Programs

Program	Semantics	Complexity
Positive DLP	minimal model (GCWA)	Π_2^P-complete
	WGCWA	co-NP-complete
	possible model (PWA)	co-NP-complete
Normal DLP	disjunctive stable model (GCWA$^\neg$)	Π_2^P-complete
	WGCWA$^\neg$	Π_2^P-complete
	possible model (PWA$^\neg$)	co-NP-complete

3.7 Discussion

In this section, we present the background of the possible model semantics and comparisons with related work.

3.7.1 Declarative Semantics

The minimal model semantics of positive disjunctive programs was firstly introduced by Minker [1982] and extended by Przymusinski [1988a] to the perfect model semantics for (locally) stratified disjunctive programs. Further extensions to normal disjunctive programs have been done in the context of the stable model semantics [Przymusinski, 1991a; Gelfond and Lifschitz, 1991] and the well-founded semantics [Ross, 1989b; Przymusinski, 1991b; Baral et al., 1992a]. As non-minimal model approaches, Ross and Topor [1988], and Rajasekar et al. [1989] have proposed the DDR and the WGCWA as a counterpart of the GCWA. However, they present only negative inference in inclusive disjunctive programs and do not provide any model theoretical meaning for such programs. Ross and Topor also suggest in their paper the usage of integrity constraints to distinguish exclusive disjunctions from inclusive ones, but they give no semantics for such programs. To characterize the meaning of inclusive disjunctive programs, Dix [1992a] presents the *weak* perfect/stationary model semantics for normal disjunctive programs without integrity constraints. However, these weak semantics have some drawbacks compared with the possible model semantics. First, the weak semantics cannot represent exclusive disjunctive programs. Second, the weak semantics do not work well in the presence of integrity constraints. For example, in Example 3.1 the weak semantics of the program is given by $\mathcal{MM}_P \cup \{M_{Horn(P)}\} = \{\{a\},\{b\},\{c\},\{a,b,c,d,e\}\}$, but as noted there, $\{a,b,c,d,e\}$ is not a model of P. Then, if we choose models satisfying the constraint

and give the meaning of P by $\{\{a\},\{b\},\{c\}\}$, it cannot represent the inclusive disjunction $a \vee b$ anymore.

To treat both exclusive and inclusive disjunctions, Ross [1989b] has proposed the *optimal* well-founded semantics which can distinguish two types of disjunctions in normal disjunctive programs. However, his semantics requires each rule to be *clarified* whether it is exclusive or inclusive, and it cannot treat a disjunctive clause containing hybrid disjunctions as presented in the introductory example. Dung [1991] has also presented a *completion* theory of negation which can distinguish two types of disjunctions in a program. However, it is defined for only positive disjunctive programs and also cannot treat hybrid disjunctions in a program. Przymusinski [1991b] suggests that his *stationary semantics* can also treat two types of disjunctions by altering the GCWA and the WGCWA during the construction of completions of disjunctions, while it seems impossible to treat disjunctive clauses containing hybrid disjunctions. Gelfond [1991] has developed an epistemic theory for disjunctive programs and provided a flexible mechanism for inferring closed world negation. His approach is based on modal logic and is different from our object-level approach.

Recently, Either et al. [1993] have introduced a circumscriptive approach for inclusive disjunctions in a first-order theory. Their *good models* provide a model theoretical counterpart of inclusive interpretations of disjunctions. In contrast to our approach, however, their *Curb* theory is defined for a first-order theory and is classical in its nature. For instance, $\{a,b\}$ is a possible model of the program $\{\ a \vee b \leftarrow,\ \ a \leftarrow\ \}$ as presented in Section 3.2.2, while they identify the above program with $\{a \leftarrow\}$ and $\{a\}$ is the unique good model. In this sense, their approach is syntax-independent and different from our syntax-dependent logic programming approach. Moreover, their Curb theory is defined for a first-order theory and its application to logic programming is limited to positive disjunctive programs.

The possible model semantics was also independently discovered by Chan [1993] under the name of the possible world semantics. Lately it was rediscovered by Decker [1992] under the name of the *sustained model semantics*. Decker and Casamayor [1993] have also shown that their sustained world assumption, which corresponds to the PWA, satisfies the properties such as *cautious monotonicity*, *cumulativity* and *rationality* in the sense of [Kraus et al., 1990]. These works have characterized the possible model semantics from different viewpoints, while they consider only positive disjunctive programs and extensions to normal disjunctive programs are not studied in the literature.

To distinguish two types of disjunctions, one may consider that instead of inserting integrity constraints, inserting *cyclic* clauses under the usual minimal model semantics is enough to interpret inclusive disjunctions. But this is not the case. Consider to

make the disjunction $a \vee b$ inclusive by adding cyclic clauses $a \leftarrow b$ and $b \leftarrow a$ to it. The resultant program now implies the equivalence $a \Leftrightarrow b$. Applying it to the introductory example, we obtain *land-animal* $\Leftrightarrow$ *aquatic*, which is of course not our intention.

We have used integrity constraints to distinguish exclusive disjunctions from inclusive ones. Then if one wishes to simulate the GCWA under the PWA, it is enough to insert integrity constraints for each exclusive disjunction. Such a simulation is discussed in [Chan, 1993].

3.7.2 Fixpoint Semantics

A fixpoint semantics for disjunctive programs has been studied by several researchers. An early approach to provide a fixpoint semantics for positive disjunctive programs was given by [Minker and Rajasekar, 1990]. In the paper, they developed a fixpoint operator which operates on *states*, sets of positive disjunctions from the extended Herbrand base. Then they showed that its fixpoint closure characterizes the minimal model semantics of positive disjunctive programs. Our fixpoint semantics is basically different from theirs in the following points. First, our fixpoint operator is designed to compute possible models as well as minimal models of disjunctive programs. Second, our fixpoint semantics is well-defined not only for positive disjunctive programs, but also for every normal disjunctive program. Third, our fixpoint construction is based on the manipulation of standard Herbrand interpretations and does not require any extension of the Herbrand base. The state based fixpoint semantics have also been developed for stratified disjunctive programs in [Rajasekar and Minker, 1989] and for normal disjunctive programs in [Baral et al., 1992a; Przymusinski, 1990b; Przymusinski, 1991b]. Reed et al. [1991] provide a different fixpoint semantics which characterizes logical consequences of a positive disjunctive program.

Fernandez and Minker [1991b] have presented a fixpoint semantics for stratified disjunctive programs using a fixpoint operator over sets of interpretations. With this fixpoint operator, they have shown that its iterative fixpoint characterizes the perfect models of a stratified disjunctive program. In [Fernandez and Minker, 1992; Fernandez et al., 1993], the result is further extended to normal disjunctive programs, in which they have developed a method for computing stable models by transforming a normal disjunctive program into a stratified disjunctive program with integrity constraints. Their approach is close to ours, however, an essential difference is that their fixpoint operator computes minimal and stable models, while ours also computes possible models. Inoue and Sakama [1992] developed yet another fixpoint semantics for computing minimal and stable models but not possible models.

Decker [1992; 1994] has also developed a fixpoint semantics of disjunctive programs. His fixpoint operator maps a disjunction of interpretations into a disjunction of interpretations, and computes *sustained models* which is equivalent to possible models. However, he provides the fixpoint semantics only for positive disjunctive programs, and its extension to normal disjunctive programs is not discussed.

Ross and Topor [1988] have given a fixpoint construction for positive disjunctive programs to characterize the semantics of the DDR, but their fixpoint closure computes the least Herbrand model of a transformed Horn program and does not characterize any model theoretical meaning of the original program.

3.7.3 Proof Procedure

Proof procedures for disjunctive programs are developed by several researchers. Fernandez et al. [Fernandez and Minker, 1991a; Fernandez et al., 1993] develop a model generation proof procedure for computing minimal and stable models of disjunctive programs using a similar program transformation to ours. Compared with their approach, our algorithm is designed for computing not only minimal/stable models but also possible models of a program, and is easily realizable in a non-deterministic or-parallel environment like [Inoue et al., 1992]. The model generation procedure presented in Section 3.5 might be considered as a variant of SATCHMO [Manthey and Bry, 1988] or MGTP [Inoue et al., 1992], but these procedures are designed to compute minimal/stable models and different from ours. For computing possible models, Chan [1993] presents a different procedure which, given a positive disjunctive program P and its model M, finds a subset of M that is also a possible model of P.

We have also presented a method of using a bottom-up procedure to evaluate queries under the possible model semantics. As an alternative approach, we can design a top-down proof procedure for the possible model semantics as follows. In Section 3.6 we have presented that possible models of a normal disjunctive program can be expressed in terms of stable models of a normal logic program by using the *pm*-transformation. This means that, using the *pm*-transformation, a top-down proof procedure for the stable model semantics of normal logic programs can also be used as a procedure for the possible model semantics of normal disjunctive programs. For instance, Eshghi and Kowalski's [1989] *abductive proof procedure* is known to be correct with respect to *call-consistent* normal logic programs. Since the *pm*-transformation preserves the call-consistency, the abductive procedure is also used as a proof procedure for the possible model semantics. For positive disjunctive programs, yet other top-down procedures are developed in [Sakama, 1989; Decker, 1992; Decker and Casamayor, 1993].

3.8 Summary

In this chapter, we have introduced the possible model semantics for positive and normal disjunctive programs, which is an alternative non-minimal model approach to the declarative semantics of disjunctive programs. The possible model semantics gives a uniform framework to treat both inclusive and exclusive disjunctions in a program, and provides a flexible negative inference mechanism compared with the previously proposed closed world assumptions. The possible model semantics was also characterized by a new fixpoint semantics of disjunctive programs.

For computing possible models, we have presented a bottom-up model generation proof procedure for positive and normal disjunctive programs. The procedure is sound and complete with respect to the possible model semantics as well as the minimal/stable model semantics in function-free range-restrieted programs. We have also shown that the possible model semantics has a computational advantage over the minimal/stable model semantics.

In normal disjunctive programs, we have defined the possible model semantics based on the stable model semantics. However, since its definition is given through the set of split normal logic programs, it is easy to construct another version of the possible model semantics based on any semantics of normal logic programs other than the stable model semantics. In this sense, the possible model semantics presented in this chapter provides a fairly general framework independent of any specific semantics. In other words, it establishes the principle of *possibilism* as a semantical counterpart of the traditional minimalism, which contributes to enrich our perspectives for commonsense reasoning in logic programming and artificial intelligence.

Chapter 4

Relating Disjunctive Logic Programs to Default Theories

In this chapter, we present the relationship between disjunctive programs and Reiter's default logic. We first point out the problem of previously proposed approaches, and propose an alternative default translation of normal disjunctive programs. The results are applied to extended disjunctive programs, and a correspondence between default logic and Gelfond et al.'s disjunctive default logic is investigated. We also address the connections between disjunctive programs and other major nonmonotonic formalisms such as Moore's autoepistemic logic and McCarthy's circumscription. The possible model semantics of disjunctive programs is also characterized by autoepistemic logic.

4.1 Introduction

Logic programming realizes a kind of default reasoning in the presence of default negation in a program. Such default reasoning is, on the other hand, known as *nonmonotonic reasoning* in artificial intelligence. Recent studies have shed light on the relationship between logic programming semantics and nonmonotonic reasoning, and it is now known that each of these areas relate to the other in a wide scope.

Default logic initially introduced by Reiter [1980] is known as one of the major formalisms of nonmonotonic reasoning in AI, and it turned out that default logic is closely related to the declarative semantics of logic programming. Bidoit and Froidevaux [1991a; 1991b] have firstly investigated the relationship between logic programming and default logic and introduced a *positivist default theory* for stratifiable and non-stratifiable logic programs. Marek and Truszczynski [1989a] have also developed transformations from logic programs to default theories, and shown a one-to-one

correspondence between the stable models of a logic program and its corresponding default extensions. The result was further extended by Gelfond and Lifschitz [1991] to logic programs with classical negation, in which they present a connection between answer sets of a program and default extensions of its corresponding default theory.

It is often said that a difficulty of Reiter's default logic arises when one considers default reasoning with disjunctive information. Using a popular example from [Poole, 1989], when we consider the default rules:

$$\frac{: lh\text{-}usable \wedge \neg lh\text{-}broken}{lh\text{-}usable}, \quad \frac{: rh\text{-}usable \wedge \neg rh\text{-}broken}{rh\text{-}usable}$$

with the disjunctive formula:

$$lh\text{-}broken \vee rh\text{-}broken,$$

they have a single extension containing both $lh\text{-}usable$ and $rh\text{-}usable$, which is unintuitive since the justifications of the defaults $\neg lh\text{-}broken$ and $\neg rh\text{-}broken$ cannot hold at the same time.

In the context of disjunctive programs, Bidoit and Hull [1986] present a one-to-one correspondence between the minimal models of a positive disjunctive program P and the extensions of a default theory which is obtained from P by adding defaults:

$$\frac{: \neg A}{\neg A}$$

for each atom A from $\mathcal{HB}_P$. In the presence of default negation in a program, Bidoit and Froidevaux [1991a] present a relationship between a stratified disjunctive program and its associated positivist default theory. However, as pointed out in this chapter, Bidoit and Froidevaux's positivist default theory contains a problem and cannot be applicable to a disjunctive program with negation even if it is stratifiable. Recently, Gelfond et al. [1991] proposed a new framework called *disjunctive default logic* which is a direct extension of Reiter's default logic. While the disjunctive default logic is closely related to the answer set semantics of extended disjunctive programs, it remains open whether there is a correspondence between Reiter's default logic and disjunctive programs in general.

In this chapter, we study the relation between disjunctive programs and default theories. We first point out the problem of Bidoit and Froidevaux's positivist default theories and propose an alternative correct default translation of normal disjunctive programs. The result is further extended to a transformation from extended disjunctive programs to default theories. We then present a connection between

default theories and disjunctive default theories through extended disjunctive programs. Furthermore, we investigate correspondences between disjunctive programs and other major nonmonotonic formalisms, circumscription and autoepistemic logic, and present methods of expressing stable models and possible models of disjunctive programs in terms of those nonmonotonic formalisms.

The rest of this chapter is organized as follows. In Section 4.2, we introduce basic notations of default logic. In Section 4.3, we point out problems of previously studied results and introduce an alternative default translation of normal disjunctive programs. In Section 4.4, we extend the results to extended disjunctive programs, and present a connection between default and disjunctive default theories. In Section 4.5, we relate disjunctive programs to autoepistemic logic and circumscription, and show an autoepistemic translation of the possible model semantics. Section 4.6 summarizes this chapter.

4.2 Default Logic

Classical first-order logic is always *monotonic*, that is, adding new axioms will never invalidate old theorems. However, this monotonic feature of classical logic is not necessarily adequate for formulating human commonsense reasoning, since we often confront situations where complete knowledge is not available, but nevertheless must draw conclusions. In such situations, we have to jump to conclusions with suitable default assumptions, and such conclusions might be revised after getting more accurate information.

Nonmonotonic logics are frameworks for such reasoning and they play important roles today as theoretical tools for commonsense reasoning in AI. Among many nonmonotonic formalisms, Reiter's *default logic* [Reiter, 1980] is known as a simple and powerful framework.

In default logic, nonmonotonic inference is presented by a *default rule*. For instance, the sentence "birds normally fly" can be represented as

$$\frac{bird(x) : fly(x)}{fly(x)}.$$

The above rule is read as "if x is a bird and it is consistent to assume that x flies, then conclude that x flies". Thus, if all we know about Tweety is that she is a bird $bird(Tweety)$, then it is concluded that she can fly $fly(Tweety)$. However, if we learn that Tweety is a penguin, and we already know that penguins cannot fly, then the

assumption $fly(Tweety)$ now becomes inconsistent and the application of the default is blocked.

In default logic, knowledge about the world is represented as a *default theory* which consists of a set of first-order formulas and a set of default rules. A set of first-order formulas represents valid but incomplete information about the world, while a set of default rules supplements the first-order formulas with the ability of reasoning with incomplete information. A default theory is formally stated as follows.

A default theory D is a set of default rules of the form:

$$\frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma} \tag{4.1}$$

where $\alpha, \beta_1, \ldots, \beta_n$ and γ are quantifier-free first-order formulas and respectively called the *prerequisite*, the *justifications* and the *consequent*.[1] A default rule (4.1) with the empty prerequisite (resp. empty justifications) is called a *prerequisite-free* (resp. *justification-free*) default. A default theory which consists of prerequisite-free (resp. justification-free) defaults is called a *prerequisite-free default theory* (resp. *justification-free default theory*).

Note here that the above definition, which is due to [Gelfond et al., 1991], is different from the original one [Reiter, 1980] in which the theory is given by the pair (D, W) of defaults and first-order formulas. As noted in [Gelfond et al., 1991], since a formula F in W is viewed as a special default with the prerequisite *true* (or empty) and the empty justification:

$$\frac{:}{F}$$

in D, both definitions are equivalent. Hence, throughout this chapter, we do not distinguish W from D, and such a special default is written by F instead of $\frac{:}{F}$. A default rule with variables is considered as a shorthand for the set of all its ground instances obtained by substituting variables with the ground terms from the language of D.

A set of formulas S is *deductively closed* if $S = Th(S)$ where Th is the deductive closure operator as usual. An extension of a default theory is defined as follows.

Definition 4.1 Let D be a default theory and E be a set of formulas. Then E is an *extension* of D if it coincides with the smallest deductively closed set of formulas E' satisfying the condition: for any ground instance of any default rule of the form (4.1) from D, if $\alpha \in E'$ and $\neg\beta_1, \ldots, \neg\beta_n \notin E$ then $\gamma \in E'$. □

[1] Here we consider quantifier-free defaults for simplicity reasons. Such a convention is also assumed in [Gelfond et al., 1991].

A default theory may have none, one or multiple extensions in general. Any extension of a default theory is *minimal*, that is, for any two extensions E and F, $E \subseteq F$ implies $E = F$. In particular, if the set of all justification-free defaults from D is inconsistent,[2] D has the unique *contradictory extension* which consists of every formula in the language of D. An extension is called *consistent* if it is not contradictory.

The following results are due to [Reiter, 1980], which present basic properties of default extensions.

Proposition 4.1 Let D be a default theory. Then E is an extension of D iff $E = \bigcup_{i=0}^{\infty} E_i$ where

$$E_0 = \{F \mid F \text{ is a first-order formula in } D\},$$
$$E_{i+1} = Th(E_i) \cup \{\gamma \mid \frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma} \in D \text{ where } \alpha \in E_i \text{ and } \neg\beta_1, \ldots, \neg\beta_n \notin E\}. \ \square$$

Proposition 4.2 Let D be a default theory and E be an extension of D. Then,

$$E = Th(\{\gamma \mid \frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma} \in D \text{ where } \alpha \in E \text{ and } \neg\beta_1, \ldots, \neg\beta_n \notin E\}). \ \square$$

Note that the converse of the above proposition does not hold in general.

Example 4.1 Let D be the default theory:

$$\{\frac{a : b}{b}, \quad \frac{b : a}{a}\}.$$

Then $E = Th(\{a, b\})$ satisfies the above equation, while the extension of D is $\emptyset$. □

A set E satisfying the equation in Proposition 4.2 is called a *weak extension* of D [Marek and Truszczynski, 1989b].

4.3 Default Translation of Normal Disjunctive Programs

In this section, we first review previously studied results on translating normal disjunctive programs into default theories. The problem of those approaches is pointed out, and an alternative correct default translation of normal disjunctive programs is presented.

[2] A set of justification-free defaults is inconsistent iff its extension is inconsistent.

4.3.1 Positivist Default Theory Revisited

To relate logic programming with default theories, Bidoit and Froidevaux [1991a] have presented a transformation from disjunctive programs to so-called *positivist default theories*. According to [Bidoit and Froidevaux, 1991a], the transformation is presented as follows.

Definition 4.2 ([Bidoit and Froidevaux, 1991a]) Let P be a normal disjunctive program. Then the *positivist default theory* D associated with P is constructed as follows:

(i) For each *not*-free clause $A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m$ in P, its corresponding first-order formula $B_1 \wedge \ldots \wedge B_m \supset A_1 \vee \ldots \vee A_l$ is in D.

(ii) Each clause containing *not* in its body $A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not B_{m+1} \wedge \ldots \wedge not B_n$ in P is transformed into the following default in D:

$$\frac{B_1 \wedge \ldots \wedge B_m : \neg B_{m+1}, \ldots, \neg B_n}{A_1 \vee \ldots \vee A_l}.$$

(iii) For each atom A in $\mathcal{HB}_P$, the following *CWA-default* is in D:

$$\frac{: \neg A}{\neg A}.$$

(iv) Nothing else is in D. □

Then, [Bidoit and Froidevaux, 1991a] claims that a positivist default theory associated with a stratified disjunctive program always has at least one extension ([Bidoit and Froidevaux, 1991a, Theorem 3.5]). Moreover,

> *([Bidoit and Froidevaux, 1991a, Theorem 4.1.3]) Let P be a stratifiable logical database. Then M is a perfect model for P iff M is a default model for its positivist default theory.*

In the above theorem, a "default model" means an Herbrand model of an extension and a "logical database" corresponds to a disjunctive program in our terminology. However, the following example shows that *there exists a stratified disjunctive program whose positivist default theory does not have any extension.*

Example 4.2 Let P be the stratified disjunctive program:

$$\{\, a \leftarrow b \wedge not\, c, \quad b \leftarrow a \wedge not\, c, \quad a \vee b \leftarrow \,\},$$

which has the perfect model $\{a, b\}$. Then consider its positivist default theory D:

$$\{\, \frac{b : \neg c}{a}, \quad \frac{a : \neg c}{b}, \quad a \vee b, \quad \frac{: \neg a}{\neg a}, \quad \frac{: \neg b}{\neg b}, \quad \frac{: \neg c}{\neg c} \,\}.$$

If we assume $E = Th(\{a, b, \neg c\})$, then $E' = Th(\{a \vee b, \neg c\})$ is the smallest deductively closed set satisfying each default in D. Since $E \neq E'$, E is not an extension. In fact, D has no extension. □

The above example shows that the result presented in [Bidoit and Froidevaux, 1991a] is incorrect. In fact, when a program contains disjunctive information as well as negation, the positivist default theory causes a problem.[3] This observation leads to the assertion that the result [Przymusinski, 1990a, Theorem 5.2], which presents the relationship between positivist default theories and the disjunctive stable model semantics, does not hold too. Since previously presented results turned out to be incorrect, we now need modification and reconstruction of theories to relate disjunctive programs with default theories.

4.3.2 Representing Normal Disjunctive Programs by Default Theories

Now we present an alternative transformation from disjunctive programs to default theories.

Definition 4.3 Let P be a normal disjunctive program. Then its *associated default theory* D_P is constructed as follows:

(i) Each clause $A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\, B_{m+1} \wedge \ldots \wedge not\, B_n$ in P is transformed into the following default in D_P:

$$\frac{: \neg B_{m+1}, \ldots, \neg B_n}{B_1 \wedge \ldots \wedge B_m \supset A_1 \vee \ldots \vee A_l}. \tag{4.2}$$

In particular, each integrity constraint: $\leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\, B_{m+1} \wedge \ldots \wedge not\, B_n$ in P is transformed into the following default in D_P:

$$\frac{: \neg B_{m+1}, \ldots, \neg B_n}{B_1 \wedge \ldots \wedge B_m \supset false}.$$

[3] According to our analysis, the proof of Lemma 3.3 in [Bidoit and Froidevaux, 1991a] seems to contain an error. However, if a disjunctive program contains no *not*, the positivist default theory reduces to the defaults presented in [Bidoit and Hull, 1986] and it works well.

(ii) For each atom A in $\mathcal{HB}_P$, the following *CWA-default* is in D_P:

$$\frac{: \neg A}{\neg A}. \tag{4.3}$$

(iii) Nothing else is in D_P. □

Notice that the difference between the positivist default theory and the associated default theory is the transformation of clauses containing default negation. That is, in our transformation, a normal disjunctive program is translated into a prerequisite-free default theory. Both transformations coincide for positive disjunctive programs.

Now several remarks are in order. Marek and Truszczynski [1989a] have developed three kinds of transformations tr_1, tr_2 and tr_3 from normal logic programs to default theories. Considering these transformations in the context of disjunctive programs, the transformation (4.2) can be regarded as an extension of their transformation tr_2 except that we are considering the CWA-default (4.3). While a transformation based upon tr_3 corresponds to the positivist default theory presented in the previous section, it has already turned out inappropriate to characterize disjunctive programs. A tr_1-based transformation translates each clause into the default:

$$\frac{B_1 \wedge \ldots \wedge B_m : \neg B_{m+1}, \ldots, \neg B_n}{A_1 \vee \ldots \vee A_l},$$

together with CWA-defaults for each atom.

The difference between tr_1 and tr_3 is that in tr_3 each *not*-free clause is transformed into a first-order formula in D, while in tr_1 it is transformed into a justification-free default in D. However, this tr_1-based transformation is also inappropriate to characterize disjunctive programs as the following example shows.

Example 4.3 Let us consider the program:

$$\{\, a \leftarrow b, \quad b \leftarrow a, \quad a \vee b \leftarrow \,\}.$$

Then, by the above tr_1-hased transformation, it is translated into the set of defaults:

$$\{\, \frac{b:}{a}, \quad \frac{a:}{b}, \quad a \vee b, \quad \frac{: \neg a}{\neg a}, \quad \frac{: \neg b}{\neg b} \,\},$$

which has no extension. □

These observations tell us that, from the viewpoint of extending three default transformations for normal logic programs in [Marek and Truszczynski, 1989a], the tr_2-based transformation is the only candidate that can be used to characterize the semantics of disjunctive programs.

Then we verify the correctness of the transformation. We first address some features of prerequisite-free default theories.

Lemma 4.3 Let D be a prerequisite-free default theory. Then E is an extension of D iff

$$E = Th(\{\gamma \mid \frac{: \beta_1, \ldots, \beta_n}{\gamma} \in D \text{ where } \neg\beta_1, \ldots, \neg\beta_n \notin E\}).$$

Proof: By Proposition 4.1, E is an extension of D iff $E = \bigcup_{i=0}^{\infty} E_i$ where

$$\begin{aligned} E_0 &= \{F \mid F \text{ is a first-order formula in } D\}, \\ E_{i+1} &= Th(E_i) \cup \{\gamma \mid \frac{: \beta_1, \ldots, \beta_n}{\gamma} \in D \text{ where } \neg\beta_1, \ldots, \neg\beta_n \notin E\}. \end{aligned}$$

Then $E_i = Th(E_i)$ for $i \geq 2$, and the result immediately follows. □

The above lemma presents that prerequisite-free default theories provide a sufficient condition to assure the converse of Proposition 4.2. That is, the notions of weak extensions and extensions coincide for prerequisite-free default theories [Marek and Truszczynski, 1989b].

The above result is further simplified as follows. Let D be a default theory and E be a set of formulas. Then, let D^E be a default theory which is obtained from D by

$$\begin{aligned} D^E = \{\frac{\alpha:}{\gamma} \mid \frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma} &\text{ is a ground instance of a default in } D \\ &\text{and } \neg\beta_1, \ldots, \neg\beta_n \notin E\} \end{aligned}$$

where D^E is called the *reduct* of D with respect to E [Gelfond et al., 1991]. Then the following property holds.

Lemma 4.4 ([Gelfond et al., 1991]) A set of formulas E is an extension of a default theory D iff E is an extension of the justification-free default theory D^E. □

From the above two lemmas, we get the following result.

Proposition 4.5 Let D be a prerequisite-free default theory. Then E is an extension of D iff $E = Th(D^E)$. □

Now we are ready to prove the main result of this section. Before that, we recall the following result for positive disjunctive programs.

Lemma 4.6 ([Bidoit and Hull, 1986; Lobo and Subrahmanian, 1992])
Let P be a positive disjunctive program. If E is a consistent extension of D_P, then $E \cap \mathcal{HB}_P$ is a minimal model of P. □

Theorem 4.7 Let P be a normal disjunctive program and D_P be its associated default theory. Then the following relations hold.

(i) If M is a stable model of P, then there is an extension E of D_P such that $M = E \cap \mathcal{HB}_P$.

(ii) If E is a consistent extension of D_P, then $M = E \cap \mathcal{HB}_P$ is a stable model of P.

Proof: (i) Suppose that M is a stable model of P and let $E = Th(M \cup \neg\overline{M})$ where $\neg\overline{M} = \{\neg A \mid A \in \mathcal{HB}_P \setminus M\}$. Then, for each clause $A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m$ in P^M, the corresponding formula $B_1 \wedge \ldots \wedge B_m \supset A_1 \vee \ldots \vee A_l$ is in ${D_P}^E$. Since M is a minimal model of P^M and ${D_P}^E = P^M \cup \{\neg A \mid A \notin M\}$, M is also a minimal model of ${D_P}^E$. Thus, $Th(M \cup \neg\overline{M}) = Th({D_P}^E)$ holds. Therefore, by Proposition 4.5, $Th(M \cup \neg\overline{M})$ is an extension of D_P, and since $Th(M \cup \neg\overline{M}) \cap \mathcal{HB}_P = M$, the result follows.

(ii) When E is a consistent extension of D_P, $E = Th({D_P}^E)$ holds by Proposition 4.5. Let $M = E \cap \mathcal{HB}_P$. Then, for each formula $B_1 \wedge \ldots \wedge B_m \supset A_1 \vee \ldots \vee A_l$ in ${D_P}^E$, the corresponding clause $A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m$ is in P^M. Since E is also an extension of ${D_P}^E$, M is a minimal model of P^M (by Lemma 4.6). Hence M is a stable model of P. □

Corollary 4.8 Let P be a normal disjunctive program. Then,

(i) P is inconsistent iff D_P has the contradictory extension.

(ii) P is consistent but incoherent iff D_P has no extension. □

The above theorem presents a one-to-one correspondence between the stable models of a normal disjunctive program and the consistent extensions of its associated default theory. The above results also reduce to the corresponding results in [Marek and Truszczynski, 1989a] for normal logic programs.

Example 4.4 ([Gelfond et al., 1991]) Let P be the program consisting of the clauses:

$$lh\text{-}usable \leftarrow not\ ab_1,$$

$$rh\text{-}usable \leftarrow not\ ab_2,$$

$$ab_1 \leftarrow lh\text{-}broken,$$

$$ab_2 \leftarrow rh\text{-}broken,$$

$$lh\text{-}broken \vee rh\text{-}broken \leftarrow .$$

These clauses are transformed into the following defaults in D_P:

$$\frac{: \neg ab_1}{lh\text{-}usable},\ \frac{: \neg ab_2}{rh\text{-}usable},\ lh\text{-}broken \supset ab_1,\ rh\text{-}broken \supset ab_2,\ lh\text{-}broken \vee rh\text{-}broken$$

with the CWA-defaults:

$$\frac{: \neg lh\text{-}broken}{\neg lh\text{-}broken},\ \frac{: \neg rh\text{-}broken}{\neg rh\text{-}broken},\ \frac{: \neg lh\text{-}usable}{\neg lh\text{-}usable},\ \frac{: \neg rh\text{-}usable}{\neg lh\text{-}usable},\ \frac{: \neg ab_1}{\neg ab_1},\ \frac{: \neg ab_2}{\neg ab_2}.$$

Then D_P has two extensions such that the sets of all atoms from them become

$$\{lh\text{-}usable, rh\text{-}broken, ab_2\} \text{ and } \{rh\text{-}usable, lh\text{-}broken, ab_1\},$$

which coincide with the stable models of P. □

The above example illustrates that Poole's paradox can be eliminated in Reiter's default theory by the effect of the CWA-defaults in the associated default theory.

4.4 Default Translation of Extended Disjunctive Programs

This section first extends the results of the previous section to extended disjunctive programs. Then we discuss a connection between default theories and disjunctive default theories recently proposed by Gelfond et al. [1991].

4.4.1 Representing Extended Disjunctive Programs by Default Theories

An extended disjunctive program is a disjunctive program containing classical negation along with default negation in the program [Gelfond and Lifschitz, 1991]. As presented in Chapter 2, an extended disjunctive program P is translated into a normal disjunctive program by considering its *positive form* P^+. Using this translation, we extend the results in the previous section to extended disjunctive programs.

Given an extended disjunctive program and its answer sets, let S^+ be a positive form of an answer set S where each negative literal $\neg A$ in S is rewritten by A' in S^+. Then the following relationship holds, which is a straightforward extension of the result for extended logic programs [Gelfond and Lifschitz, 1991, Proposition 2].

Proposition 4.9 Let P be an extended disjunctive program. Then a consistent set S is an answer set of P iff S^+ is a stable model of P^+. □

Since an extended disjunctive program reduces to a normal disjunctive program by considering its positive form, we can directly apply Definition 4.3 to give an associated default theory for an extended disjunctive program. We first rephrase Theorem 4.7 for our current use.

Lemma 4.10 Let P be an extended disjunctive program.

(i) If M is a stable model of P^+, then there is an extension E of D_{P+} such that $M = E \cap \mathcal{HB}_{P+}$.

(ii) If E is an extension of D_{P+}, then $M = E \cap \mathcal{HB}_{P+}$ is a stable model of P^+. □

We say that a consistent extension E of D_{P+} is *positively consistent* if it does not contain a pair of complementary atoms A and A'. The next theorem directly follows from the above proposition and lemma, which presents a one-to-one correspondence between the consistent answer sets of a program and the (positively) consistent extensions of its associated default theory.

Theorem 4.11 Let P be an extended disjunctive program.

(i) If S is a consistent answer set of P, then there is an extension E of D_{P+} such that $S^+ = E \cap \mathcal{HB}_{P+}$.

(ii) If E is a positively consistent extension of D_{P+}, then $S^+ = E \cap \mathcal{HB}_{P+}$ is a positive form of an answer set S of P. □

Clearly the above results reduce to the case of extended logic programs in the absence of disjunctions in a program.[4] It should be noted that when a program has no consistent answer set, we cannot apply Theorem 4.11 in a straightforward manner.

Corollary 4.12 Let P be an extended disjunctive program. If $\mathcal{L}_P$ is the unique answer set of P, then D_{P+} has no positively consistent extension. □

The converse of the above corollary does not hold in general.

Example 4.5 Let P be the extended logic program:

$$\{\, a \leftarrow not\, b, \quad \neg a \leftarrow \,\},$$

which has no answer set. In this case, its positive form P^+ becomes

$$\{\, a \leftarrow not\, b, \quad a' \leftarrow \,\},$$

and its associated default theory D_{P+} is

$$\{\, a', \quad \frac{: \neg b}{a}, \quad \frac{: \neg a}{\neg a}, \quad \frac{: \neg b}{\neg b}, \quad \frac{: \neg a'}{\neg a'} \,\},$$

which has the unique extension $Th(\{a, \neg b, a'\})$. □

To characterize a program having no consistent answer set, consider a program $P^{\mathcal{L}_P}$ which is the reduct of P with respect to $\mathcal{L}_P$. By the definition of answer sets, $\mathcal{L}_P$ is the answer set of P iff $\mathcal{L}_P$ is the answer set of $P^{\mathcal{L}_P}$. Let $P^{\mathcal{L}_P +}$ be a positive form of $P^{\mathcal{L}_P}$. Then the following result holds.

Theorem 4.13 Let P be an extended disjunctive program. Then,

(i) P has the answer set $\mathcal{L}_P$ iff $D_{P^{\mathcal{L}_P}+}$ has a consistent extension but no positively consistent extension.

(ii) P has no answer set iff either $D_{P^{\mathcal{L}_P}+}$ has a positively consistent extension but D_{P+} has no positively consistent extension, or $D_{P^{\mathcal{L}_P}+}$ has the contradictory extension. □

The results of Theorem 4.11 and 4.13 present that the answer set semantics of extended disjunctive programs is also characterized by Reiter's default theories.

[4] [Gelfond and Lifschitz, 1991] presents a default translation of extended logic programs, which is an extension of tr_1 of [Marek and Truszczynski, 1989a] and different from ours.

4.4.2 Relationship to Disjunctive Default Theory

Disjunctive default logic, recently proposed by Gelfond et al. [1991], is known as one of the extensions of Reiter's default logic, which is devised to treat default reasoning with disjunctive information. In this section, we investigate the connection between disjunctive default theories and associated default theories presented in the previous sections.

A *disjunctive default theory* Δ is a set of defaults of the form:

$$\frac{\alpha : \beta_1, \ldots, \beta_m}{\gamma_1 \mid \ldots \mid \gamma_n} \tag{4.4}$$

where α, $\beta_1, \ldots, \beta_m$, $\gamma_1, \ldots, \gamma_n$ $(m, n \geq 0)$ are quantifier-free first-order formulas and respectively called the *prerequisite*, the *justifications* and the *consequents*. The formula $\gamma_1 \mid \gamma_2$ represents a disjunction meaning "γ_1 is true or γ_2 is true", rather than "$\gamma_1 \vee \gamma_2$ is true".

An *extension* E of a disjunctive default theory is defined in the same manner as that of a default theory except that it is a minimal deductively closed set E' of formulas such that for each default rule (4.4) in Δ, if E' satisfies the prerequisite and E is consistent with the justifications, then E' is required to contain some consequent γ_i $(1 \leq i \leq n)$ rather than the disjunction itself.

Given an extended disjunctive program P, each clause

$$L_1 \vee \ldots \vee L_l \leftarrow L_{l+1} \wedge \ldots \wedge L_m \wedge not\ L_{m+1} \wedge \ldots \wedge not\ L_n$$

in P is translated into the following disjunctive default in its *associated disjunctive default theory* Δ_P:

$$\frac{L_{l+1} \wedge \ldots \wedge L_m : \neg L_{m+1}, \ldots, \neg L_n}{L_1 \mid \ldots \mid L_l}. \tag{4.5}$$

Note here that any CWA-default is not included in Δ_P. The following proposition is due to [Gelfond et al., 1991] which presents the relation between an extended disjunctive program and its associated disjunctive default theory.

Proposition 4.14 Let P be an extended disjunctive program and Δ_P be its associated disjunctive default theory. Then a consistent set S is an answer set of P iff S is the set of all literals from an extension of Δ_P. □

In the previous section, we have presented the relationship between extended disjunctive programs and default theories. Then the next results follows from Theorem 4.11, Theorem 4.13, and Proposition 4.14. Recall here that S^+ is a positive form of an answer set S.

Theorem 4.15 Let P be an extended disjunctive program.

(i) If E_Δ is a consistent extension of Δ_P and $S = E_\Delta \cap \mathcal{L}_P$, then there is an extension E of D_{P^+} such that $S^+ = E \cap \mathcal{HB}_{P^+}$.

(ii) If E is a positively consistent extension of D_{P^+} and $S^+ = E \cap \mathcal{HB}_{P^+}$, then there is an extension E_Δ of Δ_P such that $S = E_\Delta \cap \mathcal{L}_P$. □

Corollary 4.16 Let P be an extended disjunctive program. Then,

(i) Δ_P has the contradictory extension iff either $D_{p\mathcal{L}_P{}^+}$ has a consistent extension but no positively consistent extension, or $D_{p\mathcal{L}_P{}^+}$ has the contradictory extension.

(ii) Δ_P has no extension iff $D_{p\mathcal{L}_P{}^+}$ has a positively consistent extension but D_{P^+} has no positively consistent extension. □

The above results bridge the gap between disjunctive default theories and Reiter's default theories in terms of extended disjunctive programs.

In [Gelfond et al., 1991], the difficulty of expressing disjunctive information in Reiter's default theory is discussed using some examples. However, we have already seen that Poole's paradox is eliminated by considering the CWA-defaults in its associated default theory (Example 4.4). The following examples, which are also given in [Gelfond et al., 1991] to differentiate each formalism, present that we do not lose any information under Reiter's default theory in the presence of disjunctive information.

Example 4.6 Let Δ_P be the disjunctive default theory:

$$\{\ a \equiv b, \quad a \mid b\ \}.$$

Then the corresponding default theory

$$D_P = \{\ a \equiv b, \quad a \vee b, \quad \frac{: \neg a}{\neg a}, \quad \frac{: \neg b}{\neg b}\ \}$$

has the unique extension $Th(\{a, b\})$ which is equivalent to the extension of Δ_P. □

Example 4.7 Let Δ_P be the disjunctive default theory:

$$\{\ a \mid b, \quad \frac{a :}{b}, \quad \frac{: \neg a}{c}\ \}.$$

Then the corresponding default theory

$$D_P = \{\ a \vee b, \quad a \supset b, \quad \frac{: \neg a}{c}, \quad \frac{: \neg a}{\neg a}, \quad \frac{: \neg b}{\neg b}, \quad \frac{: \neg c}{\neg c}\ \}$$

has the unique extension $Th(\{\neg a, b, c\})$ where $Th(\{\neg a, b, c\} \cap \mathcal{HB}_P) = Th(\{b, c\})$ coincides with the unique extension of Δ_P. □

It remains open whether there is a general correspondence between the disjunctive default theory and the default theory.[5] However, the results presented in this section show that Reiter's default theory has the same expressiveness as the disjunctive default theory to characterize the stable model semantics and the answer set semantics of normal and extended disjunctive programs.

4.5 Connections with Autoepistemic Logic and Circumscription

In this section, we consider connections between disjunctive programs and other two representative nonmonotonic formalisms. The one is Moore's autoepistemic logic, and the other is McCarthy's circumscription.

4.5.1 Autoepistemic Logic

Autoepistemic logic [Moore, 1985] is a modal non-monotonic logic which is developed as a reconstruction of *nonmonotonic logic* by [McDermott and Doyle, 1980].

In autoepistemic logic, nonmonotonic reasoning is achieved as reasoning with an agent's own belief. For instance, the bird-fly sentence is represented as

$$bird(x) \wedge \neg \mathsf{L} \neg fly(x) \supset fly(x)$$

where L is a modal *belief* operator. The above formula is read as "if x is a bird and there is no evidence to believe that x does not fly, then x flies".

Thus, if we know the fact $bird(Tweety)$ and there is no reason to believe the opposite fact $\neg fly(Tweety)$, then $fly(Tweety)$ is concluded.

An autoepistemic theory is defined in a similar way to a first-order theory except that its language includes modal formulas built from modal operator L and usual first-order connectives, with the restriction that quantification into the scope of the modal operator is not allowed.[6]

An expansion of an autoepistemic theory is defined as follows.

[5] A recent study [Eiter and Gottlob, 1993b] indicates that both default theories and disjunctive default theories are at the same second level of the polynomial hierarchy. Therefore, "there must exist a polynomial transformation from reasoning tasks in disjunctive default theories to analogous reasoning tasks in default theories" (G. Gottlob, private communication).

[6] Thus, free variables appearing in an autoepistemic formula are viewed as representing its ground instances.

Definition 4.4 Let AE be an autoepistemic theory. Then its *expansion* E is defined as

$$E = Th(AE \cup \{\mathsf{L}\phi \mid \phi \in E\} \cup \{\neg \mathsf{L}\phi \mid \phi \notin E\}). \quad \square \tag{4.6}$$

An autoepistemic expansion E satisfies the following conditions:

(i) $E = Th(E)$,

(ii) $A \in E$ iff $\mathsf{L}A \in E$, and

(iii) $A \notin E$ iff $\neg \mathsf{L}A \in E$ (where E is consistent).

A correspondence between autoepistemic logic and logic programming is firstly studied by Gelfond [1987] for the perfect model semantics of stratified logic programs. The result is extended to the stable model semantics of normal logic programs in [Gelfond and Lifschitz, 1988].

We present an autoepistemic translation of normal disjunctive programs using the relation between default theories and autoepistemic theories. It is known that the extensions of a default theory are related to the expansions of an autoepistemic theory [Konolige, 1988; Marek and Truszczynski, 1989b]. Marek and Truszczynski [1989b] have shown that there is a one-to-one correspondence between a weak extension of a default theory and an expansion of its corresponding autoepistemic theory. Since weak extensions coincide with extensions in prerequisite-free default theories, the above result implies that the default translation of disjunctive programs presented in Section 4.3 is also rephrased in the context of autoepistemic logic. That is, in Definition 4.3 (i), instead of transforming each clause in a program into the corresponding default rule, we can transform each clause into the following autoepistemic formula:

$$B_1 \wedge \ldots \wedge B_m \wedge \neg \mathsf{L}B_{m+1} \wedge \ldots \wedge \neg \mathsf{L}B_n \supset A_1 \vee \ldots \vee A_l, \tag{4.7}$$

and instead of the CWA-defaults in (ii), we have the *CWA-formula*:

$$\neg \mathsf{L}A \supset \neg A. \tag{4.8}$$

In this way, the autoepistemic theory AE_P associated with a disjunctive program P is defined. Then the following result holds.

Theorem 4.17 Let P be a consistent normal disjunctive program and AE_P be its associated autoepistemic theory. Then M is a stable model of P iff E is an expansion of AE_P and $M = E \cap \mathcal{HB}_P$. $\square$

The above theorem provides an autoepistemic characterization of disjunctive programs. Such an autoepistemic translation is also presented in [Przymusinski, 1990a] in the context of the 3-valued stable model semantics. Using the same techniques presented in the previous sections, we can also provide autoepistemic characterizations of extended disjunctive programs and associated disjunctive default theories. An alternative autoepistemic translation for extended disjunctive programs is presented in [Lifschitz and Schwarz, 1993; Chen, 1993].

4.5.2 Circumscription

Circumscription [McCarthy, 1980; 1986] is also known as one of the popular frameworks of nonmonotonic reasoning in AI. Different from default logic and autoepistemic logic, circumscription is still within classical logic but augmented with an ability of nonmonotonic reasoning.

Using the bird-fly example again, let us consider the first-order formula:

$$\forall x\ bird(x) \wedge \neg abnormal(x) \supset fly(x).$$

Then, if $bird(Tweety)$ is the all known facts about Tweety, by minimizing the extension of the predicate *abnormal* (in this case the empty set), circumscription concludes $fly(Tweety)$. Else if there is an evidence that Tweety is an abnormal bird $abnormal(Tweety)$, $fly(Tweety)$ is not derived any more.

Several variations of circumscription have been proposed so far. In what follows, we use *predicate circumscription* originally proposed by McCarthy [1980].

Given a first-order theory T and a set of predicates $\Pi = \{p_1, \ldots, p_n\}$ from T, the circumscription of T with respect to Π is defined as the second-order axiom that

$$Circ(T; \Pi) \equiv T(p_1', \ldots, p_n') \wedge \bigwedge_{i=1}^{n} (\forall \mathbf{x}\ p_i'(\mathbf{x}) \supset p_i(\mathbf{x})) \supset \bigwedge_{i=1}^{n} (\forall \mathbf{x}\ p_i(\mathbf{x}) \supset p_i'(\mathbf{x})) \quad (4.9)$$

where each p_i' is a predicate variable with the same arities as p_i.

The above schema means that for any theory $T(p_1', \ldots, p_n')$ obtained from T by replacing each p_i with p_i', there is no theory which has smaller extensions of $p_1, \ldots, p_n$ with respect to T.

Predicates in Π are said to be *minimized*, while the rest of predicates in the language of T are said to be *fixed*. Any model of $Circ(T; \Pi)$ is called a Π-*minimal* model of T, in which extensions of each predicate from Π are minimized with fixed interpretations for the rest of predicates.

Circumscription is also closely related to logic programming semantics. Lifschitz [1985] showed a connection between predicate circumscription and the CWA. The result is further extended to stratified logic programs [Lifschitz, 1988], and normal logic programs [Lifschitz, 1989; Lin and Shoham, 1992; Yuan and You, 1993]. For positive disjunctive programs, circumscription coincides with the minimal model semantics if no fixed predicate exists. The perfect model semantics of stratified disjunctive programs is also characterized by *prioritized circumscription* [Przymusinski, 1988a]. Gelfond et al. [1989] study various forms of closed world assumptions in terms of circumscription.

In the following, we characterize the disjunctive stable model semantics of normal disjunctive programs by circumscription.

For a normal disjunctive program P, let P_L be a first-order theory which is obtained from P by replacing each *not* A in P by $\neg LA$, where LA is a new atom meaning A *is believed*.[7] Then P_L is a set of formulas of the same form as (4.7) except that each modal formula $\neg \mathsf{L} B_i$ is replaced by a first-order literal $\neg LB_i$. Given a theory P_L, its Herbrand base is defined as $\mathcal{HB}_P \cup \{LA \mid A \in \mathcal{HB}_P\}$, and an Herbrand interpretation is defined as a subset of the Herbrand base. We restrict our attention to Herbrand models of the theory P_L, since we are interested in a semantic relationship between P and P_L. Also such a restriction has an effect to incorporate both the *domain closure assumption* and the *unique name assumption* into P_L [Bossu and Siegel, 1985].

Let Π be the set of all predicates appearing in the language of P. Then circumscription $Circ(P_L; \Pi)$ represents that circumscribing the predicates Π in P_L with the fixed predicates $L\Pi$, where $L\Pi = \{Lp \mid p \in \Pi\}$. In the following, for $M \subseteq \mathcal{HB}_P$, we write $LM = \{LA \mid A \in M\}$, and $L\Pi \equiv \Pi$ means $\bigwedge_{p_i \in \Pi} \forall \mathbf{x}\ Lp_i(\mathbf{x}) \equiv p_i(\mathbf{x})$.

Let us consider a propositional theory ${P_L}^{LM}$ which is obtained from the ground instance of the theory P_L by deleting (i) each formula which has a negative literal $\neg LA$ in its antecedent such that $LA \in LM$, and (ii) all negative literals $\neg LA$ in the antecedents of the remaining formulas. Then the following lemma holds.

Lemma 4.18 Let P be a normal disjunctive program. Then $M \cup LM$ is an Herbrand model of $Circ({P_L}^{LM}; \Pi) \wedge L\Pi \equiv \Pi$ iff $M \cup LM$ is an Herbrand model of $Circ(P_L; \Pi) \wedge L\Pi \equiv \Pi$.

Proof: Suppose that $M \cup LM$ is an Herbrand model of $Circ({P_L}^{LM}; \Pi) \wedge L\Pi \equiv \Pi$. Then, for each ground formula $B_1 \wedge \ldots \wedge B_m \supset A_1 \vee \ldots \vee A_l$

[7] The meaning of LA is the same as $\mathsf{K}A$ in Chapter 3, but here we use the notation LA to compare it with an autoepistemic formula $\mathsf{L}A$.

in ${P_L}^{LM}$, $\{B_1, \ldots, B_m\} \subseteq M$ implies $A_i \in M$ for some i $(1 \leq i \leq l)$. In this case, there is a corresponding ground formula $B_1 \wedge \ldots \wedge B_m \supset A_1 \vee \ldots \vee A_l \vee LB_{m+1} \vee \ldots \vee LB_n$ from P_L such that $\{B_1, \ldots, B_m\} \subseteq M$ and $A_i \in M$ hold. Since M is Π-minimal, $M \cup LM$ is also an Herbrand model of $Circ(P_L; \Pi) \wedge L\Pi \equiv \Pi$.

Conversely, suppose that $M \cup LM$ is an Herbrand model of $Circ(P_L; \Pi) \wedge L\Pi \equiv \Pi$. Then, for each ground formula $B_1 \wedge \ldots \wedge B_m \supset A_1 \vee \ldots \vee A_l \vee LB_{m+1} \vee \ldots \vee LB_n$ from P_L, $\{B_1, \ldots, B_m\} \subseteq M$ implies either (i) $A_i \in M$ for some i $(1 \leq i \leq l)$ or (ii) $LB_j \in LM$ for some j $(m+1 \leq j \leq n)$. In case of (i), when $LB_j \notin LM$, there is a corresponding ground formula $B_1 \wedge \ldots \wedge B_m \supset A_1 \vee \ldots \vee A_l$ in ${P_L}^{LM}$ such that $\{B_1, \ldots, B_m\} \subseteq M$ implies $A_i \in M$. In case of (ii), there is no corresponding ground formula in ${P_L}^{LM}$. In each case, $M \cup LM$ is also a model of ${P_L}^{LM}$. Since M is Π-minimal, $M \cup LM$ is an Herbrand model of $Circ({P_L}^{LM}; \Pi) \wedge L\Pi \equiv \Pi$. □

Now we present the theorem which provides a connection between disjunctive programs and circumscription.

Theorem 4.19 Let P be a normal disjunctive program. Then M is a stable model of P iff $M \cup LM$ is an Herbrand model of $Circ(P_L; \Pi) \wedge L\Pi \equiv \Pi$.

Proof: M is a stable model of P
iff M is a minimal model of P^M
iff M is an Herbrand model of $Circ(P^M; \Pi)$
iff M is an Herbrand model of $Circ({P_L}^{LM}; \Pi)$
iff $M \cup LM$ is an Herbrand model of $Circ({P_L}^{LM}; \Pi) \wedge L\Pi \equiv \Pi$
iff $M \cup LM$ is an Herbrand model of $Circ(P_L; \Pi) \wedge L\Pi \equiv \Pi$ (by Lemma 4.18).
□

The above theorem generalizes the corresponding results for normal logic programs [Lifschitz, 1989; Lin and Shoham, 1992; Yuan and You, 1993], and also provides a method of characterizing extended disjunctive programs and associated disjunctive default theories in terms of circumscription.[8]

[8]For normal disjunctive programs, a similar result is also reported in [Lin and Shoham, 1992] without proof.

4.5.3 Characterizing Possible Model Semantics

We finally discuss the issue of characterizing the possible model semantics by nonmonotonic formalisms.

We have already seen that possible models of normal disjunctive programs are translated into stable models of normal logic programs by the *pm*-transformation in Section 3.6. Since stable models of normal logic programs can be expressed by the nonmonotonic frameworks as presented in the previous sections, possible models are also expressed by each nonmonotonic framework via stable models.

However, the *pm*-transformation introduces extra atoms like A_i''s, which are not included in the language of the original program. Hence nonmonotonic translations of possible models through the *pm*-transformation also include those extra-language formulas, which does not precisely coincide with the original meaning of the program.

Then, in this section we consider a possibility to express possible models directly in terms of each nonmonotonic formalism. For this purpose, we can first exclude (disjunctive) default logic and circumscription. This is because those formalisms are based on the principle of minimality, that is, default extensions and circumscribed models are always minimal. Thereby the remaining candidate is autoepistemic logic.

Fortunately, autoepistemic expansions are not necessarily minimal with respect to first-order formulas. For instance, the autoepistemic theory $\{ La \supset a \}$ has two expansions: one containing a and La, while the other containing $\neg La$ but neither a nor $\neg a$. Historically, non-minimal expansions are considered as *anomalous* expansions, and efforts have been done to eliminate such expansions [Konolige, 1988]. However, as shown below, non-minimal expansions are useful to express possible models in an autoepistemic theory.

Let us consider the autoepistemic theory $AE = \{ La \supset a, \quad Lb \supset b \}$, which has four expansions containing the sets, $E_1 = \{a, La, \neg Lb\}$, $E_2 = \{b, Lb, \neg La\}$, $E_3 = \{a, b, La, Lb\}$, and $E_4 = \{\neg La, \neg Lb\}$, respectively. Then we can observe that the first three expansions correspond to the possible models of the program $P = \{ a \vee b \leftarrow \}$.

Rewriting AE by $AE' = \{a \vee \neg La, \quad b \vee \neg Lb\}$ will help to understand the correspondence. Each formula in AE' presents that an atom is true or not believed, which exactly characterizes interpretations of split programs of P. That is, E_1, E_2, and E_3 express epistemic interpretations of split programs $P_1 = \{a \leftarrow\}$, $P_2 = \{b \leftarrow\}$, and $P_3 = \{a \leftarrow, \quad b \leftarrow\}$, respectively. On the other hand, since there is no split program corresponding to E_4, we add the formula $La \vee Lb$ to AE to remove this expansion from AE. This formula presents that when $a \vee b$ is true in P, one of the disjuncts is believed.

Formally, expressing possible models in autoepistemic logic is achieved as follows.

Definition 4.5 Let P be a normal disjunctive program. Then its associated *pm-autoepistemic theory* AE_P^{pm} is constructed as follows:

(i) Each disjunctive clause $A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\ B_{m+1} \wedge \ldots \wedge not\ B_n$ from P is transformed into the following autoepistemic formulas in AE_P^{pm}:

$$B_1 \wedge \ldots \wedge B_m \wedge \neg \mathsf{L} B_{m+1} \wedge \ldots \wedge \neg \mathsf{L} B_n \supset A_i \vee \neg \mathsf{L} A_i \quad \text{for } i = 1, \ldots, l, \tag{4.10}$$

$$B_1 \wedge \ldots \wedge B_m \wedge \neg \mathsf{L} B_{m+1} \wedge \ldots \wedge \neg \mathsf{L} B_n \supset \mathsf{L} A_1 \vee \ldots \vee \mathsf{L} A_l. \tag{4.11}$$

In particular, each normal clause $A_l \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\ B_{m+1} \wedge \ldots \wedge not\ B_n$ from P is transformed into the following autoepistemic formula in AE_P^{pm}:

$$B_1 \wedge \ldots \wedge B_m \wedge \neg \mathsf{L} B_{m+1} \wedge \ldots \wedge \neg \mathsf{L} B_n \supset A_l.$$

(ii) Nothing else is in AE_P^{pm}. □

Note here that AE_P^{pm} contains no CWA-formula (4.8).

The first formula (4.10) represents that if the antecedent of the formula is true, A_i is true or not believed. The second formula (4.11) represents that if the antecedent is true, at least one of the disjuncts is believed. Recall that an autoepistemic expansion E satisfies the conditions that $\neg \mathsf{L} A_i \in E$ implies $A_i \notin E$ and $\mathsf{L} A_i \in E$ implies $A_i \in E$.

Now we have the following result.

Theorem 4.20 Let P be a consistent normal disjunctive program and AE_P^{pm} be its associated *pm*-autoepistemic theory. Then M is a possible model of P iff E is an expansion of AE_P^{pm} and $M = E \cap \mathcal{HB}_P$.

Proof: Let M be a possible model of P. Then there is a split program P' of P such that M is a stable model of P'. Suppose that each ground disjunctive clause $A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\ B_{m+1} \wedge \ldots \wedge not\ B_n$ in P is replaced with the split clauses: $A_i \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\ B_{m+1} \wedge \ldots \wedge not\ B_n$ in P', where $A_i \in S$ for some non-empty subset S of $\{A_1, \ldots, A_l\}$. In this case, it is easy to see that the corresponding autoepistemic theory AE_P^{pm} has an expansion E such that $A_i \in M$ iff $A_i \in E \cap \mathcal{HB}_P$, and $A_i \notin M$ iff $\neg \mathsf{L} A_i \in E$. Hence, the result follows. The converse is also shown in the same manner. □

4.6 Summary

This chapter has presented the relationship between disjunctive programs and default theories.

We first pointed out the problem of Bidoit and Froidevaux's positivist default theories, and developed an alternative correct default translation of normal disjunctive programs. It was shown a one-to-one correspondence between the stable models of a disjunctive program and the default extensions of its associated default theory. We also extended the results to default translation of extended disjunctive programs and the answer set semantics. The results indicate that Reiter's default theory is as expressive as Gelfond et al.'s disjunctive default theory to characterize the semantics of disjunctive programs.

We finally presented the connections between disjunctive programs and autoepistemic logic, and circumscription. The possible model semantics of disjunctive programs was also characterized using non-minimal feature of autoepistemic expansions.

Chapter 5

Equivalence between Disjunctive and Abductive Logic Programs

Abductive logic programming is a recently proposed framework which enhances logic programming by supplying the ability of abductive reasoning in AI. In this chapter, we consider such an abductive logic programming framework and reveal its close relationship to disjunctive programs. We show that the generalized stable model semantics of abductive logic programs are viewed as the possible model semantics of disjunctive programs, and vice versa. We also demonstrate that abductive disjunctive programs do not increase expressiveness of disjunctive programs. Interrelations between various semantics of disjunctive and abductive logic programs are discussed in terms of computational complexity.

5.1 Introduction

Abduction is a form of hypothetical reasoning and is widely used today in various AI problems such as diagnosis and planning. Abduction is a reasoning for hypothetical generation from a given observation, and provides a weak kind of non-deductive inference as a tool for commonsense reasoning.

In logic programming, abduction is realized in the framework of *abductive logic programming*. The framework was firstly proposed by Eshghi and Kowalski [1989], in which they gave an abductive interpretation of negation as failure in normal logic programs. They showed a one-to-one correspondence between the stable models of a normal logic program and the extensions of its associated abductive framework. Kakas and Mancarella [1990] extended their framework to abductive logic programs containing *abducibles* which represent not necessarily default negation but positive

hypothetical facts. They also introduced the *generalized stable model semantics* as a theoretical framework of such programs. Further extensions of abductive logic programming have been studied by several researchers in the last few years [Kakas et al., 1992].

Comparing between disjunctive programs and abductive logic programs, both frameworks enhance logic programming by supplying the ability of reasoning with incomplete information. It is achieved in disjunctive programs by reasoning with disjunctive information, while in abductive logic programs by reasoning with hypotheses. Disjunctive programs and abductive logic programs have been independently developed so far and have different syntax and semantics from each other. However, in disjunctive programs, each disjunction is considered to represent knowledge about possible alternative beliefs, and such beliefs can also be regarded as a kind of hypotheses. In abductive logic programs, on the other hand, each candidate hypothesis is examined whether it is adopted or not, and this situation can be considered as meta-level disjunctive knowledge that either a hypothesis is true or not. Thus, each formalism appears to deal with very similar problems from different viewpoints. Then the question naturally arises whether there is any formal correspondence between these two frameworks.

There are some studies which can be related to the above question. Dung [1992a] presents a program transformation from acyclic disjunctive programs to normal logic programs under the stable model semantics and uses Eshghi and Kowalski's abductive proof procedure for such programs. However, Dung's transformation is restricted to acyclic disjunctive programs and not applicable in general. Console et al. [1991] characterize abduction using Clark's completion technique in abductive logic programs. They show that abductive solutions are obtained by deduction in the only-if part of the completed abductive program which is viewed as a kind of disjunctive program, but the technique is applicable only to acyclic abductive programs. Inoue and Sakama [1993] present a program transformation from abductive logic programs to disjunctive programs under the stable model semantics and use a bottom-up model generation proof procedure for computing abduction. While their transformation is fairly general, it is a one-way transformation from abductive logic programs to disjunctive programs.

In this chapter, we investigate a general correspondence between disjunctive programs and abductive logic programs. For the part from abductive logic programs to disjunctive programs, we show that the generalized stable models of an abductive logic program are characterized by the possible models of the transformed disjunctive program. Conversely, from disjunctive programs to abductive logic programs,

we show that the possible models of a disjunctive program are exactly the generalized stable models of the transformed abductive logic program. Moreover, if the disjunctive stable model semantics is taken as the underlying semantics instead of the possible model semantics, it is unlikely that disjunctive programs can be efficiently expressed in terms of the generalized stable model semantics. It is also shown that abductive disjunctive programs can be expressed by abductive logic programs under the possible model semantics.

The rest of this chapter is organized as follows. In Section 5.2, we introduce the notion of abductive logic programming. In Section 5.3, we present program transformations between abductive logic programs and disjunctive programs. It is shown that the generalized stable models of an abductive logic program are characterized by the possible models of the transformed disjunctive program, and vice versa. In Section 5.4, we introduce abductive disjunctive programs and present their translation into disjunctive programs. It is shown that abductive logic programs are as expressive as abductive disjunctive programs under the possible model semantics. Section 5.5 discusses the relation between disjunctive programs and abductive logic programs from the computational complexity viewpoint. Section 5.6 summarizes this chapter.

5.2 Abductive Logic Programming

Abduction is firstly introduced by Peirce [1932] who characterized three distinguished forms of reasoning, abduction, induction, and deduction. Abduction is a non-deductive inference and is formally presented as follows.

Given a theory T and an *observation* O, *abduction* is defined as an inference of an *explanation* E such that

$$T \cup E \models O \text{ where } T \cup E \text{ is consistent.} \tag{5.1}$$

Thus, abduction can be thought of as a form of hypothetical reasoning which produces, with background knowledge T, hypothetical sentences E that are sufficient to account for O.

Abductive logic programming is a form of such abductive frameworks in which T is given as a logic program.

An *abductive logic program* is a pair $\langle P, \mathcal{A} \rangle$ where P is a normal logic program and $\mathcal{A}$ is a finite set of atoms called the *abducibles*.[1]

[1] We slightly modified the original definition of [Kakas and Mancarella, 1990] by including integrity constraints in a program and considering abducible atoms instead of abducible predicates. Here, an abducible containing variables is identified with its ground instances.

Abducibles are pre-specified sentences, which represent candidate hypotheses used for explanations. Such a specification is needed to get "best" explanations since there might be many candidate explanations which can deduce a given observation from the condition (5.1).

A declarative semantics of abductive logic programs is given by Kakas and Mancarella [1990] who extended the stable model semantics of normal logic programs to the *generalized stable model semantics* of abductive logic programs. Similar extensions are done in [Gelfond, 1990; Inoue, 1991] in the context of extended logic programs.

Let $\langle P, \mathcal{A} \rangle$ be an abductive logic program and E be a subset of $\mathcal{A}$. An interpretation I is a *generalized stable model* of $\langle P, \mathcal{A} \rangle$ if I is a stable model of the normal logic program $P \cup E$. A generalized stable model I is called *$\mathcal{A}$-minimal* if there is no generalized stable model J such that $J \cap \mathcal{A} \subset I \cap \mathcal{A}$. Clearly, ($\mathcal{A}$-minimal) generalized stable models coincide with stable models if $\mathcal{A} = \emptyset$.

Let $\langle P, \mathcal{A} \rangle$ be an abductive logic program and O be an atom which represents *observation*. Then a set $E \subseteq \mathcal{A}$ is an *explanation* of O if there is a generalized stable model I of $\langle P, \mathcal{A} \rangle$ such that I satisfies O and $E = I \cap \mathcal{A}$. An explanation E of O is *minimal* if no $E' \subset E$ is an explanation of O.

Note that the problem of finding explanations is essentially equivalent to the problem of finding generalized stable models since E is a (minimal) explanation of O with respect to $\langle P, \mathcal{A} \rangle$ iff I is a ($\mathcal{A}$-minimal) generalized stable model of $\langle P \cup \{ \leftarrow not\, O \}, \mathcal{A} \rangle$ such that $I \cap \mathcal{A} = E$.

Also note that without loss of generality an observation is assumed to be a non-abducible ground atom. This is because if O is an abducible, its explanations trivially contain O. Else if $O(\mathbf{x})$ contains a tuple of free variables $\mathbf{x}$, we can introduce a new proposition O which is considered as an observation in the program $P \cup \{ O \leftarrow O(\mathbf{x}) \}$. Else if multiple observations are given like that $O_1, \ldots, O_m$ are observed and $O_{m+1}, \ldots, O_n$ are not observed, they are also realized by introducing a clause $O \leftarrow O_1 \wedge \ldots \wedge O_m \wedge not\, O_{m+1} \wedge \ldots \wedge not\, O_n$ into P and computing explanations of O [Inoue and Sakama, 1993].

Example 5.1 Let $\langle P, \mathcal{A} \rangle$ be an abductive logic program such that

$$P = \{\ p(x) \leftarrow q(x) \wedge not\ r(x),\quad q(x) \leftarrow s(x),\quad q(x) \leftarrow t(x)\ \}$$

and $\mathcal{A} = \{\ s(x),\ t(b)\ \}$. Then, for a given observation $O = p(a)$, the ($\mathcal{A}$-minimal) generalized stable model $I = \{\ p(a),\ q(a),\ s(a)\ \}$ of $\langle P, \mathcal{A} \rangle$ satisfies O and its (minimal) explanation is $E = I \cap \mathcal{A} = \{\ s(a)\ \}$. Here, I is also the unique generalized stable model of $\langle P \cup \{ \leftarrow not\, p(a) \}, \mathcal{A} \rangle$. □

5.3 Connections between Disjunctive and Abductive Logic Programs

In this section, we characterize disjunctive programs in terms of abductive logic programs, and vice versa. Then we reveal close relationships between the generalized stable models of abductive logic programs and the possible models of disjunctive programs.

5.3.1 Generalized Stable Models are Possible Models

We first present a program transformation from abductive logic programs to disjunctive programs, then show that the generalized stable models of an abductive logic program can be expressed by the possible models of the transformed disjunctive program.

In an abductive logic program, each candidate hypothesis is either assumed or not. Such a situation is naturally expressed by disjunctions in a program.

Definition 5.1 Let $\langle P, \mathcal{A} \rangle$ be an abductive logic program. Then its *dlp-transformation* is defined by a normal disjunctive program $dlp(\langle P, \mathcal{A} \rangle)$ which is obtained from P by adding the following disjunctive clauses for each abducible $A \in \mathcal{A}$:

$$A \vee \varepsilon \leftarrow \tag{5.2}$$

where ε is an atom not appearing elsewhere in P. □

The intuitive meaning of the *dlp*-transformation is that when an abducible A is assumed in an abductive logic program $\langle P, \mathcal{A} \rangle$, the corresponding disjunct A is chosen from (5.2) in the transformed disjunctive program $dlp(\langle P, \mathcal{A} \rangle)$. Else when A is not assumed, the newly introduced atom ε is chosen from (5.2). Thus the *dlp*-transformation specifies meta-level knowledge representing whether each abducible is assumed or not.

Now we express the generalized stable model semantics in terms of $dlp(\langle P, \mathcal{A} \rangle)$. Let I be a possible model of a normal disjunctive program P. We say that I is *$\mathcal{A}$-minimal* if there is no possible model J of P such that $J \cap \mathcal{A} \subset I \cap \mathcal{A}$. In the following, an atom A is identified with the unit clause $A \leftarrow$ in E.

Theorem 5.1 Let $\langle P, \mathcal{A} \rangle$ be an abductive logic program. Then,

(i) $I \setminus \{\varepsilon\}$ is a generalized stable model of $\langle P, \mathcal{A} \rangle$ iff I is a possible model of $dlp(\langle P, \mathcal{A} \rangle)$.

(ii) $I \setminus \{\varepsilon\}$ is an $\mathcal{A}$-minimal generalized stable model of $\langle P, \mathcal{A} \rangle$ iff I is an $\mathcal{A}$-minimal possible model of $dlp(\langle P, \mathcal{A} \rangle)$.

Proof: (i) Let I' be a generalized stable model of $\langle P, \mathcal{A} \rangle$. Then I' is a stable model of $P \cup E$ for some E from $\mathcal{A}$. Now let us consider the transformed disjunctive program $dlp(\langle P, \mathcal{A} \rangle)$. Then there is a split program P' of $dlp(\langle P, \mathcal{A} \rangle)$ such that for each disjunctive clause (5.2), $A \leftarrow$ is in P' if $A \in E$; $\varepsilon \leftarrow$ is in P', otherwise. When $\varepsilon \leftarrow$ is in P', $I' \cup \{\varepsilon\}$ is a stable model of P' and also a possible model of $dlp(\langle P, \mathcal{A} \rangle)$. Else when $\varepsilon \leftarrow$ is not in P', I' is a stable model of P' and also a possible model of $dlp(\langle P, \mathcal{A} \rangle)$. Hence the result of only-if part follows.

Conversely, when I is a possible model of $dlp(\langle P, \mathcal{A} \rangle)$, it is a stable model of some split program P' of $dlp(\langle P, \mathcal{A} \rangle)$. Let E be the set of all split clauses included in P'. Then I is a stable model of $P \cup E$. Since $E \setminus \{\varepsilon \leftarrow\}$ consists of instances from $\mathcal{A}$, $I \setminus \{\varepsilon\}$ is a generalized stable model of $\langle P, \mathcal{A} \rangle$.

(ii) The result directly follows from (i) and the definitions of $\mathcal{A}$-minimal generalized stable models/$\mathcal{A}$-minimal possible models. □

Corollary 5.2 Let $\langle P, \mathcal{A} \rangle$ be an abductive logic program. Then, for a given observation O, there is a (minimal) explanation E of O iff there is an ($\mathcal{A}$-minimal) possible model I of $dlp(\langle P, \mathcal{A} \rangle)$ satisfying O and $I \cap \mathcal{A} = E$. □

Example 5.2 Let $\langle P, \mathcal{A} \rangle$ be an abductive logic program such that

$$\begin{aligned} P \;=\; \{\; & \textit{wet-shoes} \leftarrow \textit{wet-grass} \wedge \textit{not driving-car}, \\ & \textit{wet-grass} \leftarrow \textit{rained}, \\ & \textit{wet-grass} \leftarrow \textit{sprinkler-on} \;\}, \end{aligned}$$

and $\mathcal{A} = \{\; \textit{rained},\ \textit{sprinkler-on} \;\}$. Then,

$$dlp(\langle P, \mathcal{A} \rangle) = P \cup \{\; \textit{rained} \vee \varepsilon \leftarrow,\quad \textit{sprinkler-on} \vee \varepsilon \leftarrow \;\}$$

which has the five possible models:

$$\{\textit{rained}, \textit{sprinkler-on}, \textit{wet-grass}, \textit{wet-shoes}\},$$

$$\{\varepsilon\},$$

$$\{\varepsilon, \textit{rained}, \textit{wet-grass}, \textit{wet-shoes}\},$$

$$\{\varepsilon, \textit{sprinkler-on}, \textit{wet-grass}, \textit{wet-shoes}\},$$

$$\{\varepsilon, \textit{rained}, \textit{sprinkler-on}, \textit{wet-grass}, \textit{wet-shoes}\}.$$

Thus, the generalized stable models of $\langle P, \mathcal{A} \rangle$ coincide with the sets which are obtained by removing ε from each possible model. In particular, $\{\varepsilon\}$ is the $\mathcal{A}$-minimal possible model and it corresponds to the $\mathcal{A}$-minimal generalized stable model $\emptyset$ of $\langle P, \mathcal{A} \rangle$. □

The result of this section indicates that abductive logic programs are also considered as disjunctive programs. In the next section, we present that the converse is also the case.

5.3.2 Possible Models are Generalized Stable Models

As presented in the introduction, indefinite information in disjunctive programs is viewed as possible hypotheses in a program. Then it is natural to represent disjuncts in terms of abducibles in an abductive logic program. However, the problem is that disjunctive clauses possibly have conditions in their bodies, while abductive logic programs introduced in Section 5.2 lack the ability of expressing assumptions with preconditions. Then our first task is to extend the framework of abductive logic programs to possibly include such hypothetical rules.

An abductive logic program considering in this section is a pair $\langle P, \mathcal{C} \rangle$ where P is a normal logic program and $\mathcal{C}$ is a finite set of normal clauses called the *abducible rules*. The abducible rule intuitively means that if the rule is abduced then it is used for inference together with the background knowledge from P. In this sense, abductive logic programs presented in the previous sections are considered as a special case where each abducible rule has the empty precondition. The generalized stable model semantics of such an extended framework is defined as follows.

Definition 5.2 Let $\langle P, \mathcal{C} \rangle$ be an abductive logic program and F be a subset of $\mathcal{C}$. An interpretation I is a *generalized stable model* of $\langle P, \mathcal{C} \rangle$ if it is a stable model of the normal logic program $P \cup F$. □

The generalized stable model introduced above is a direct extension of the one presented in the previous sections, and it reduces to the usual notion when $\mathcal{C} = \mathcal{A}$.

Next we provide a program transformation which translates normal disjunctive programs into abductive logic programs. For a normal disjunctive program P, we define $P = disj(P) \cup \overline{disj}(P)$ where $disj(P)$ is the set of all disjunctive clauses from P and $\overline{disj}(P)$ is the set of all normal clauses and integrity constraints from P. In the following, Γ denotes the conjunction in the body of a clause.

Definition 5.3 Given a normal disjunctive program P, let us consider the set of normal clauses

$$\mathcal{C} = \{\ A_i \leftarrow \Gamma \mid A_1 \vee \ldots \vee A_l \leftarrow \Gamma \in disj(P) \text{ and } 1 \leq i \leq l\ \} \tag{5.3}$$

and the integrity constraints

$$IC = \{\ \leftarrow \Gamma \wedge not\ A_1 \wedge \ldots \wedge not\ A_l \mid A_1 \vee \ldots \vee A_l \leftarrow \Gamma \in disj(P)\ \}. \tag{5.4}$$

Then we define the *alp-transformation* of P by $alp(P) = \langle\, \overline{disj}(P) \cup IC, \mathcal{C}\,\rangle$. □

The intuitive meaning of the *alp*-transformation is that each disjunctive clause in a program is replaced with a set of abducible rules (5.3) in $\mathcal{C}$. The integrity constraints (5.4) in IC impose the condition that at least one of disjuncts is chosen as an abducible whenever the body of a disjunctive clause is true. In this way, by the *alp*-transformation each disjunctive clause is rewritten by a set of abducible rules.

Now we present the relationship between the possible models of a normal disjunctive program P and the generalized stable models of the transformed abductive logic program $alp(P)$.

Theorem 5.3 Let P be a normal disjunctive program. Then I is a possible model of P iff I is a generalized stable model of $alp(P)$.

Proof: Let I be a possible model of P. Then there is a split program P' of P such that I is a stable model of P'. Suppose that each ground disjunctive clause $C^k:\ A_1 \vee \ldots \vee A_{l_k} \leftarrow \Gamma_k$ from P is replaced with the split clauses in $C^k_S = \{A_i \leftarrow \Gamma_k \mid A_i \in S\}$ in P' where S is a non-empty subset of $\{A_1, \ldots, A_{l_k}\}$. Then I is a stable model of $\overline{disj}(P) \cup \bigcup_k C^k_S$. Since $\bigcup_k C^k_S$ consists of instances from $\mathcal{C}$ and I satisfies integrity constraints IC, I is also a generalized stable model of $alp(P)$.

Conversely, let I be a generalized stable model of $alp(P)$. Then I is a stable model of $P \cup F$ where F is a subset of $\mathcal{C}$. For each normal clause $A_i \leftarrow \Gamma$ in F, there is a corresponding disjunctive clause $C:\ A_1 \vee \ldots \vee A_l \leftarrow \Gamma$ in $disj(P)$ such that $1 \leq i \leq l$. Also, since I satisfies integrity constraints IC, when I satisfies Γ, at least one normal clause $A_i \leftarrow \Gamma$ is included in F. In this case, there is a split program P' of P in which each ground instance of a disjunctive clause C is split into a corresponding ground instance of a normal clause $A_i \leftarrow \Gamma$. Thus I is also a stable model of P', hence a possible model of P. □

Example 5.3 ([Chan, 1993]) Let

$$\begin{aligned} P \;=\; \{\ & violent \vee psychopath \leftarrow suspect, \\ & dangerous \leftarrow violent \wedge psychopath, \\ & suspect \leftarrow\ \}. \end{aligned}$$

Then, $alp(P) = \langle\, \overline{disj}(P) \cup IC, \mathcal{C}\,\rangle$ where

$$\begin{aligned} \overline{disj}(P) \cup IC \;=\; \{\ & dangerous \leftarrow violent \wedge psychopath, \\ & suspect \leftarrow, \\ & \leftarrow suspect \wedge not\ violent \wedge not\ psychopath\ \}, \\ \mathcal{C} \;=\; \{\ & violent \leftarrow suspect,\quad psychopath \leftarrow suspect\ \}. \end{aligned}$$

Thus, $alp(P)$ has three generalized stable models:

$$\{suspect,\ violent\},$$

$$\{suspect, psychopath\},$$

$$\{suspect,\ violent,\ psychopath,\ dangerous\},$$

which coincide with the possible models of P. □

Note that in the above example there is no minimal model of P containing *dangerous*. By contrast, $alp(P)$ has a generalized stable model in which *dangerous* is true, which corresponds to a possible model in which the disjunction is inclusively true.

The abductive logic programming framework presented in this section is also introduced by Inoue [1991] in the context of the *knowledge system* for extended logic programs. He also shows that an abductive logic program $\langle\, P, \mathcal{C}\,\rangle$ can be translated into a semantically equivalent usual abductive logic program $\langle\, P, \mathcal{A}\,\rangle$. Given an abductive logic program $\langle\, P, \mathcal{C}\,\rangle$, let us consider a program P' which is obtained from P by including the clause:

$$A \leftarrow A' \wedge \Gamma$$

for each abducible rule $A \leftarrow \Gamma$ in $\mathcal{C}$. Here A' is a newly introduced atom not appearing elsewhere in P and is uniquely associated with each A. Also let $\mathcal{A}'$ be a set of abducibles which consists of every newly introduced atom A'. Then he proves that there is a one-to-one correspondence between the generalized stable models of $\langle\, P, \mathcal{C}\,\rangle$ and the generalized stable models of $\langle\, P', \mathcal{A}'\,\rangle$. This fact implies that the possible models of a normal disjunctive program are also expressed by the generalized stable models of a usual abductive logic program.

5.4 Abductive Disjunctive Programs

This section extends a framework of abductive logic programs to abductive disjunctive programs, and discusses their correspondence to disjunctive programs and abductive logic programs.

5.4.1 Generalized Disjunctive Stable Models and Possible Models

Abductive disjunctive programs are disjunctive programs with abducibles. The definition of an abductive disjunctive program $\langle P, \mathcal{A} \rangle$ is the same as an abductive logic program except that P is a normal disjunctive program. For a given set $E \subseteq \mathcal{A}$, an interpretation I is a *generalized disjunctive stable model* of $\langle P, \mathcal{A} \rangle$ if I is a disjunctive stable model of the normal disjunctive program $P \cup E$. On the other hand, I is a *generalized possible model* of $\langle P, \mathcal{A} \rangle$ if I is a possible model of the normal disjunctive program $P \cup E$. A generalized disjunctive stable model (resp. generalized possible model) I is *$\mathcal{A}$-minimal* if there is no generalized disjunctive stable model (resp. generalized possible model) J such that $J \cap \mathcal{A} \subset I \cap \mathcal{A}$.

The above definitions are direct extensions of the previously proposed notions. In fact, generalized disjunctive stable models (resp. generalized possible models) reduce to disjunctive stable models (resp. possible models) in normal disjunctive programs with $\mathcal{A} = \emptyset$, and both generalized disjunctive stable models and generalized possible models reduce to generalized stable models in abductive logic programs.

A difference between generalized disjunctive stable models and generalized possible models is illustrated in the following example.

Example 5.4 Let $\langle P, \mathcal{A} \rangle$ be an abductive disjunctive program such that

$$P = \{\, a \vee b \leftarrow c, \quad d \leftarrow a \wedge b \,\}$$

and $\mathcal{A} = \{\, c \,\}$. Then, $\emptyset$, $\{c, a\}$, $\{c, b\}$, $\{c, a, b, d\}$ are all generalized possible models, while $\{c, a, b, d\}$ is not a generalized disjunctive stable model. Thus, for a given observation $O = d$, it has an explanation c under the generalized possible models, while no explanation is available under the generalized disjunctive stable models. □

In this way, the generalized possible model semantics can provide explanations which come from inclusive disjunctions, while the generalized disjunctive stable model semantics cannot in general.

5.4.2 Generalized Possible Models are Generalized Stable Models

Abductive disjunctive programs are generalization of disjunctive programs. However, in this section we present a somewhat unexpected result that abductive disjunctive programs do not increase expressive power of disjunctive programs. We show this fact by introducing a translation from abductive disjunctive programs into disjunctive programs.

For an abductive disjunctive program $\langle P, \mathcal{A} \rangle$, we define its *dlp-transformation* $dlp(\langle P, \mathcal{A} \rangle)$ in the same manner as presented in Definition 5.1. Then the following results hold.

Theorem 5.4 Let $\langle P, \mathcal{A} \rangle$ be an abductive disjunctive program. Then,

(i) $I \setminus \{\varepsilon\}$ is a generalized possible model of $\langle P, \mathcal{A} \rangle$ iff I is a possible model of $dlp(\langle P, \mathcal{A} \rangle)$.

(ii) $I \setminus \{\varepsilon\}$ is an $\mathcal{A}$-minimal generalized possible model of $\langle P, \mathcal{A} \rangle$ iff I is an $\mathcal{A}$-minimal possible model of $dlp(\langle P, \mathcal{A} \rangle)$.

Proof: Similar to the proof of Theorem 5.1. □

The above theorem, together with Theorem 5.3, implies the following result.

Corollary 5.5 Let $\langle P, \mathcal{A} \rangle$ be an abductive disjunctive program. Then $I \setminus \{\varepsilon\}$ is a generalized possible model of $\langle P, \mathcal{A} \rangle$ iff I is a generalized stable model of $alp(dlp(\langle P, \mathcal{A} \rangle))$. □

By Theorem 5.4, normal disjunctive programs are as expressive as abductive disjunctive programs under the possible model semantics. Moreover, Corollary 5.5 presents that abductive disjunctive programs can be expressed even by abductive logic programs under the generalized possible model semantics.

Next we consider the corresponding relations under the disjunctive stable model semantics.

Given an abductive disjunctive program $\langle P, \mathcal{A} \rangle$, its *$dlp_{st}$-transformation* is defined as a normal disjunctive program $dlp_{st}(\langle P, \mathcal{A} \rangle)$ which is obtained from P by adding the following disjunctive clauses for each abducible $A \in \mathcal{A}$:

$$A \vee \varepsilon_A \leftarrow \tag{5.5}$$

where ε_A is an atom not appearing elsewhere in P, and is uniquely associated with each A.

Note here that different from the *dlp*-transformation presented in Definition 5.1, the unique ε_A is associated with each A in (5.5). This modification is needed to assure the existence of a disjunctive stable model corresponding to any generalized stable model of $\langle P, \mathcal{A} \rangle$ (see Example 5.5 below).

Let us call I an $\mathcal{A}$-minimal disjunctive stable model if it is a disjunctive stable model such that $I \cap \mathcal{A}$ is minimal. Then the following results hold.

Theorem 5.6 Let $\langle P, \mathcal{A} \rangle$ be an abductive disjunctive program. Then,

(i) $I \setminus \{\varepsilon_A\}$ is a generalized disjunctive stable model of $\langle P, \mathcal{A} \rangle$ iff I is a disjunctive stable model of $dlp_{st}(\langle P, \mathcal{A} \rangle)$.

(ii) $I \setminus \{\varepsilon_A\}$ is an $\mathcal{A}$-minimal generalized disjunctive stable model of $\langle P, \mathcal{A} \rangle$ iff I is an $\mathcal{A}$-minimal disjunctive stable model of $dlp_{st}(\langle P, \mathcal{A} \rangle)$.

Proof: (i) Let I' be a generalized disjunctive stable model of $\langle P, \mathcal{A} \rangle$. Then I' is a disjunctive stable model of $P \cup E$ for some E from $\mathcal{A}$. Now let us consider the transformed disjunctive program $dlp_{st}(\langle P, \mathcal{A} \rangle)$. Then there is a disjunctive stable model I of $dlp_{st}(\langle P, \mathcal{A} \rangle)$ such that for each disjunctive clause $A \vee \varepsilon_A \leftarrow$, $A \in I$ iff $A \in I'$, and $\varepsilon_A \in I$ iff $A \notin I'$. Hence the result follows. The converse is also shown in the same manner.

(ii) The result holds from (i) and the definition of $\mathcal{A}$-minimal (generalized) disjunctive stable models. □

Corollary 5.7 Let $\langle P, \mathcal{A} \rangle$ be an abductive logic program. Then $I \setminus \{\varepsilon_A\}$ is a generalized stable model of $\langle P, \mathcal{A} \rangle$ iff I is a disjunctive stable model of $dlp_{st}(\langle P, \mathcal{A} \rangle)$. □

Example 5.5 Let $\langle P, \mathcal{A} \rangle$ be an abductive disjunctive program such that $P = \emptyset$ and $\mathcal{A} = \{a, b\}$. Then, $\emptyset$, $\{a\}$, $\{b\}$, and $\{a, b\}$ are the generalized disjunctive stable models of $\langle P, \mathcal{A} \rangle$. On the other hand,

$$dlp_{st}(\langle P, \mathcal{A} \rangle) = \{\ a \vee \varepsilon_a \leftarrow,\quad b \vee \varepsilon_b \leftarrow\ \},$$

and $\{\varepsilon_a, \varepsilon_b\}$, $\{a, \varepsilon_b\}$, $\{b, \varepsilon_a\}$, and $\{a, b\}$ are the corresponding disjunctive stable models of $dlp_{st}(\langle P, \mathcal{A} \rangle)$.

Note here that if we do not distinguish ε_a and ε_b, $\{a, \varepsilon_b\}$ and $\{b, \varepsilon_a\}$ do not become disjunctive stable models of $dlp_{st}(\langle P, \mathcal{A} \rangle)$. □

The above theorem presents that normal disjunctive programs are also as expressive as abductive disjunctive programs under the disjunctive stable model semantics.[2] However, in contrast to the case of the possible model semantics, abductive disjunctive programs cannot be efficiently reduced to abductive logic programs under the generalized disjunctive stable model semantics. We verify this fact in the next section from the computational complexity viewpoint.

5.5 Discussion

In this section, we discuss the computational aspects of disjunctive and abductive logic programs. Throughout the section, programs are assumed to be propositional programs.

When abductive logic programs do not contain default negation, Selman and Levesque [1990] and Eiter and Gottlob [1992] show that the decision problem of the existence of explanations for a given observation in an abductive Horn program is NP-complete. In other words, in an abductive Horn program, deciding whether there is a generalized stable model satisfying an observation is NP-complete.

Inoue [1991] and Satoh and Iwayama [1991] show that an abductive logic program can be translated into a semantically equivalent normal logic program. For an abductive logic program $\langle P, \mathcal{A} \rangle$, consider a normal logic program obtained from P by adding the following clauses for each abducible A in $\mathcal{A}$:

$$A \leftarrow not\ A',$$

$$A' \leftarrow not\ A,$$

where A' is a newly introduced atom not appearing elsewhere in P and is uniquely associated with each A. Then these authors show that there is a one-to-one correspondence between the generalized stable models of $\langle P, \mathcal{A} \rangle$ and the stable models of the transformed normal logic program. Since it is known that the set-membership problem under the stable model semantics is NP-complete [Marek and Truszczynski, 1991a], the above polynomial-time translation implies that deciding whether there is a generalized stable model satisfying a given observation is also NP-complete.[3]

[2] Similar results are presented in [Inoue and Sakama, 1993; 1994] using different transformations.

[3] More precisely, the generalized stable models include the stable models as a special case, then its set-membership problem is NP-hard. Since the polynomial-time transformation translates the decision problem for a generalized stable model into the corresponding problem for a stable model which is in NP, the membership in NP also follows.

Table 5.1: Comparison of Computational Complexity

Program	Semantics	Complexity
Abductive LP	Horn Abduction	NP-complete
	Generalized Stable Model	NP-complete
Normal DLP	Possible Model	NP-complete
	Disjunctive Stable Model	Σ_2^P-complete
Abductive DLP	Generalized Possible Model	NP-complete
	Generalized Disjunctive Stable Model	Σ_2^P-complete

From the results presented in Section 5.4, generalized possible models can be efficiently translated into possible models. Since we have already seen in Section 3.6 that the set-membership problem under the possible model semantics is NP-complete, the corresponding decision problem for generalized possible models is also NP-complete.

On the other hand, we have also shown in Section 5.4 that abductive disjunctive programs are reducible to normal disjunctive programs under the disjunctive stable model semantics. Since the set-membership problem under the disjunctive stable model semantics is Σ_2^P-complete [Eiter and Gottlob, 1993a], deciding whether there is a generalized disjunctive stable model satisfying a given observation is also Σ_2^P-complete. These results are summarized in Table 5.1.

The above complexity measures verify the results of this chapter that the generalized stable model semantics of abductive logic programs can be expressed in terms of the possible model semantics of normal disjunctive programs by a polynomial-time transformation, and vice versa. Moreover, we can observe that *there is no efficient way to express the disjunctive stable model semantics in terms of the generalized stable model semantics unless the polynomial hierarchy collapses*. This observation extends the fact that disjunctive stable models cannot be expressed by stable models of a normal logic program in polynomial time [Eiter and Gottlob, 1993b]. Also we can observe that when considering to extend the framework of abductive logic programs to abductive disjunctive programs, *the generalized possible model semantics enables us to extend the framework without increasing computational complexity, while this is not the case for the generalized disjunctive stable model semantics.*

The possible model semantics is originally introduced in order to provide a flexible mechanism for closed world assumptions in disjunctive programs. However, the results of this chapter reveal that the possible model semantics is also useful to characterize abductive logic programs. Moreover, in Section 3.6 we have shown that normal

disjunctive programs are reducible to normal logic programs under the possible model semantics. Since abductive disjunctive programs are reducible to normal disjunctive programs, we can conclude that adding disjunctions and abducibles does not increase the expressive power of normal logic programs under the possible model semantics.

5.6 Summary

This chapter has investigated formal correspondences between disjunctive programs and abductive logic programs.

We first presented program transformations between abductive logic programs and disjunctive programs. It was shown that the generalized stable models of an abductive logic program are characterized by the possible models of the transformed normal disjunctive program, and vice versa.

Next we showed that disjunctive programs are as expressive as abductive disjunctive programs. Moreover, normal disjunctive programs, abductive logic programs, and abductive disjunctive programs are all equivalent under the possible model semantics. On the other hand, we have argued that expressing the disjunctive stable model semantics in terms of generalized stable models is most unlikely possible in polynomial time.

The results of this chapter indicate that disjunctive programs and abductive logic programs are just different ways of looking at the same problem if we choose the appropriate semantics. Also the usefulness of the possible model semantics was verified not only for disjunctive programs but also for abductive logic programs.

Chapter 6

Handling Inconsistency in Disjunctive Logic Programs

An extended disjunctive program is an extension of a normal disjunctive program, which contains classical negation along with default negation. In the presence of explicit negation, however, an extended disjunctive program possibly becomes inconsistent, and classical logic programming semantics are of no use in such inconsistent programs. In this chapter, we present theoretical frameworks for possibly inconsistent disjunctive programs. To achieve this goal, we introduce *paraconsistent* semantics for extended disjunctive programs, which can distinguish inconsistent information from other information in a program. These semantics are based on lattice-structured multi-valued logics, and are characterized by a fixpoint semantics of extended disjunctive programs. The proposed paraconsistent semantics are used for reasoning in inconsistent programs.

6.1 Introduction

Representing and reasoning with incomplete information in a program is one of the central issues in recent studies of logic programming. *Extended disjunctive programs* introduced by Gelfond and Lifschitz [1991] provide a fairly general framework for that purpose. An extended disjunctive program can specify incomplete information by using classical negation as well as disjunctions in a program. In the presence of such explicit negation in a program, however, an extended disjunctive program possibly becomes contradictory, since negative consequences are allowed in the program. In [Gelfond and Lifschitz, 1991], a declarative semantics of extended disjunctive programs is given by the notion of *answer sets*, which is a generalization of stable models

of normal disjunctive programs. However, the problem of the answer set semantics is that the answer set becomes trivial in an inconsistent program and implies every formula from the program. This is also the case for most of the traditional logic programming semantics in which local inconsistency might spoil the whole program. Practically speaking, when we build a large-scale knowledge base in logic programming framework, inconsistent information as well as incomplete information is likely to happen in the knowledge base. In such a knowledge base, a piece of contradictory information would make the whole program inconsistent, but still the program may contain meaningful information which is not affected by the local inconsistency.

Paraconsistent logic is a logic which is not destructive in the presence of inconsistent information. In this logic, the contradictory statement $A \wedge \neg A$ does not deduce an arbitrary formula, hence would not trivialize the whole theory. In this regard, paraconsistent logic can localize inconsistent information in a theory and serves as a useful inference tool in artificial intelligence. Historically, paraconsistent logic has been developed in the area of philosophical logic [Arruda, 1980], and a formal framework for inconsistent theories was given by da Costa [1974]. Applications of paraconsistent logic to logic programming have also been investigated by several researchers. Blair and Subrahmanian [1989] firstly introduced a framework of *paraconsistent logic programming*. They extended Fitting's three-valued semantics of logic programming [Fitting, 1985] and developed a theory for possibly inconsistent logic programs using Belnap's *four-valued logic* [Belnap, 1975]. The result was generalized by Subrahmanian [1992] to programs containing disjunctive information. Recently, the paraconsistent logic programming framework was further extended to treat default negation along with explicit negation in a program [Pimentel and Rodi, 1991; Wagner, 1991a; Kifer and Lozinskii, 1992; Grant and Subrahmanian, 1992]. However, in the context of extended disjunctive programs, a suitable paraconsistent extension of the answer set semantics has not been studied in the literature.

In this chapter, we present declarative semantics of possibly inconsistent extended disjunctive programs. The disjunctive stable model semantics and the possible model semantics of normal disjunctive programs are extended to the corresponding paraconsistent semantics for extended disjunctive programs. The proposed paraconsistent semantics are based on lattice-structured multi-valued logics, and are characterized by the fixpoint semantics of extended disjunctive programs. We also present applications of the paraconsistent semantics for reasoning in inconsistent programs.

The rest of this chapter is organized as follows. In Section 6.2, we first present the paraconsistent minimal and possible model semantics for positive extended disjunctive programs. The results are generalized in Section 6.3 to the paraconsistent stable and possible model semantics for extended disjunctive programs. In Section 6.4, a fixpoint semantics for extended disjunctive programs is presented to characterize the paraconsistent semantics of extended disjunctive programs. In Section 6.5, we discuss the issue of reasoning in inconsistent programs. The notions of preferred stable models, suspicious stable models, and semi-stable models are introduced as variants of the paraconsistent stable model semantics. Section 6.6 addresses comparisons with related work, and Section 6.7 summarizes this chapter.

6.2 Paraconsistent Semantics for Positive Extended Disjunctive Programs

In this section, we first consider positive extended disjunctive programs, that is, extended disjunctive programs containing no default negation.

6.2.1 Multi-valued Logic

To formalize the semantics of possibly inconsistent logic programs, multi-valued logics are often used instead of the traditional two-valued logic. Then we start from introducing such a new logic and define its model theory.

The set of truth values of *four-valued logic* is defined as $IV = \{\mathbf{t}, \mathbf{f}, \top, \bot\}$, in which $\mathbf{t}$, $\mathbf{f}$, $\top$, $\bot$ are propositions in the language of a program and respectively denote *true*, *false*, *contradictory*, and *undefined*. The set of truth values IV makes a complete lattice under the ordering $\preceq$ such that $\bot \preceq \mathbf{x} \preceq \top$ for $\mathbf{x} \in \{\mathbf{t}, \mathbf{f}\}$ (Figure 6.1). Such a lattice is also known as Belnap's four-valued logic [Belnap, 1975].[1]

In extended disjunctive programs, negative literals have the same status as positive literals, then it is natural to consider the Herbrand base of a program P as the set of all ground literals $\mathcal{L}_P$ from the language of P. For simplicity, we assume that programs do not contain the reserved propositions from IV and $IV \not\subset \mathcal{L}_P$.[2]

Let I be a subset of $\mathcal{L}_P$. An *interpretation* of a positive extended disjunctive program P is defined as a function $I : \mathcal{L}_P \to IV$ such that for each literal $L \in \mathcal{L}_P$,

$$I(L) = \begin{cases} \mathbf{t} & \text{if } L \in I \text{ and } \neg L \notin I, \\ \mathbf{f} & \text{if } \neg L \in I \text{ and } L \notin I, \\ \top & \text{if both } L \in I \text{ and } \neg L \in I, \\ \bot & \text{otherwise.} \end{cases}$$

[1] Note that the order considering here is the so-called *knowledge ordering*, while there is the alternative *truth ordering* in the context of *bilattices* [Ginsberg, 1988; Fitting, 1991].

[2] This assumption is not essential and can be removed, but this issue is not discussed here.

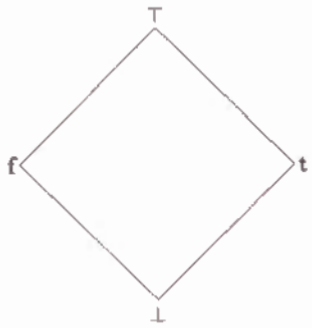


Figure 6.1: Four-valued lattice IV

Note that $I(L) = \mathbf{t}$ iff $I(\neg L) = \mathbf{f}$, $I(L) = \top$ iff $I(\neg L) = \top$, and $I(L) = \bot$ iff $I(\neg L) = \bot$.

In this chapter, when no coufusion arises, we identify a set of literals I with its interpretation $I(L)$ for each $L \in I$. For instance, we identify $I = \{L\}$ with $I(L) = \mathbf{t}$; $I = \{L, \neg L\}$ with $I(L) = \top$; $I = \emptyset$ with $I(L) = \bot$ for any $L \in \mathcal{L}_P$, and so on.

Satisfaction of each clause in a program is inductively defined as follows.

Definition 6.1 Let P be a positive extended disjunctive program and I be an interpretation. Then,

1. For any literal $L \in \mathcal{L}_P$,
   (a) $I \models L$ iff $\mathbf{t} \preceq I(L)$,
   (b) $I \models \neg L$ iff $\mathbf{f} \preceq I(L)$.
2. For any disjunction of ground literals $F = L_1 \vee \ldots \vee L_n$,
   $I \models F$ iff $I \models L_i$ for some i $(1 \leq i \leq n)$.
3. For any conjunction of ground literals $G = L_1 \wedge \ldots \wedge L_n$,
   $I \models G$ iff $I \models L_i$ for every i $(1 \leq i \leq n)$.
4. For any ground clause $C = F \leftarrow G$, $I \models C$ iff $I \models F$ or $I \not\models G$.
   In particular, $I \models \leftarrow G$ iff $I \not\models G$, and $I \models F \leftarrow$ iff $I \models F$. □

An interpretation I is a *model* of a positive extended disjunctive program P if I satisfies every ground clause from P. The ordering $\preceq$ on truth values is also defined between interpretations. For interpretations I and J, $I \preceq J$ iff $I(L) \preceq J(L)$ for any $L \in \mathcal{L}_P$. The orderings $\succeq$, $\prec$, $\succ$ are defined in the usual way. Note that when we identify a set of literals with its interpretation, $I \preceq J$ iff $I \subseteq J$. A model I is *minimal* if there is no model J such that $J \prec I$. A model I is *least* if $I \preceq J$ for every model J. To distinguish terms from standard logic programming, a minimal/least model is also called a *paraconsistent minimal/least model* (shortly, *p-minimal/p-least model*). A *consistent model* is a model I such that $I(L) \neq \top$ for any $L \in \mathcal{L}_P$, otherwise I is an *inconsistent model*. A positive extended disjunctive program is called *consistent* if it has a consistent model, otherwise it is called *inconsistent*.

Proposition 6.1 If a positive extended disjunctive program has a model, it has at least one p-minimal model.

> *Proof:* Let us consider a decreasing sequence of models $I_1 \supseteq I_2 \supseteq \ldots$ and their greatest lower bound $I = \bigcap_{i \geq 1} I_i$. Then, for each ground clause $F \leftarrow G$ from a positive extended disjunctive program P, if $I \models G$, $I_i \models G$ for any $i \geq 1$. In this case, since each I_i is a model of P, F is not empty and $I_i \models F$ holds for any $i \geq 1$. Thus $I \models F$. Hence, I is also a model of P, and by definition it is a p-minimal model. □

Proposition 6.2 A consistent positive extended disjunctive program has a consistent p-minimal model.

> *Proof:* When a positive extended disjunctive program P is consistent, it has a consistent model I by definition. If I is not minimal, there is a p-minimal model J such that $J \prec I$ by Proposition 6.1. Then, $I(L) \neq \top$ for any $L \in \mathcal{L}_P$ implies $J(L) \neq \top$ for any $L \in \mathcal{L}_P$. □

Corollary 6.3 If a positive extended logic program has a model, it has the unique p-least model. In particular, a consistent positive extended logic program has the consistent p-least model. □

Example 6.1 Let P be the program:

$$\{\, a \vee b \leftarrow, \quad \neg a \leftarrow, \quad \neg b \leftarrow, \quad c \leftarrow \,\}.$$

Then P has two p-minimal models $\{a, \neg a, \neg b, c\}$ and $\{b, \neg a, \neg b, c\}$. □

Note that the above program is inconsistent and the classical minimal model semantics makes the program trivial, while the p-minimal models retain truth information about c that is not affected by the inconsistency.

Remark:

1. As presented in Chapter 2, the meanings of the clauses $\leftarrow L$ and $\neg L \leftarrow$ are different in extended disjunctive programs. In our multi-valued setting, this is also the case. In fact, $I(L) = \bot$ is a model of the first clause, while it is not a model of the second clause. Such a difference is due to the non-contrapositive feature of the connective $\leftarrow$ in extended disjunctive programs.

2. Corresponding to the above fact, the program $\{L \leftarrow, \quad \neg L \leftarrow\}$ has a model $I(L) = \top$ while $\{L \leftarrow, \quad \leftarrow L\}$ has no model. That is, we consider any interpretation meaningless if it does not satisfy integrity constraints. However, it is easy to construct a paraconsistent theory for integrity violation if desired.

6.2.2 Paraconsistent Possible Model Semantics

The possible model semantics for positive disjunctive programs is extended to positive extended disjunctive programs in a straightforward manner.

Given a positive extended disjunctive program P, a *split program* is defined as a ground positive extended logic program obtained from P by replacing each ground disjunctive clause $C : L_1 \vee \ldots \vee L_l \leftarrow L_{l+1} \wedge \ldots \wedge L_m$ with the following ground extended clauses (called *split clauses*):

$$L_i \leftarrow L_{l+1} \wedge \ldots \wedge L_m \quad \text{for every } L_i \in S$$

where S is some non-empty subset of $\{L_1, \ldots, L_l\}$. Then a *p-possible model* of P is defined as the p-least model of any split program of P.

By definition, any p-possible model is a model of P.

Example 6.2 Let P be the program:

$$\{\ a \vee b \leftarrow, \quad c \leftarrow a \wedge b, \quad \neg c \leftarrow\ \}.$$

Then P has the three p-possible models $\{a, \neg c\}$, $\{b, \neg c\}$ and $\{a, b, c, \neg c\}$. □

Note here that if P contains the integrity constraint $\leftarrow c$ instead of the clause $\neg c \leftarrow$, the positive disjunctive program has two p-possible models $\{a\}$ and $\{b\}$.

The following properties are direct consequences from Propositions 6.1 and 6.2.

Proposition 6.4 When a positive extended disjunctive program has a model, it has at least one p-possible model. □

Proposition 6.5 A consistent positive extended disjunctive program has a consistent p-possible model. □

As for the relation to p-minimal models, the following property holds.

Proposition 6.6 Any p-minimal model is a minimal p-possible model, and vice versa. □

6.3 Paraconsistent Semantics for Extended Disjunctive Programs

This section extends the results presented in the previous section to extended disjunctive programs in general.

6.3.1 Paraconsistent Stable Model Semantics

We first extend the model theory given for positive extended disjunctive programs in the previous section to extended disjunctive programs containing default negation.

In an extended disjunctive program, the notion of interpretations and satisfaction of literals and clauses are defined in the same manner as in Definition 6.1 except that for each formula *not* L, we include the additional statements:

- $I \models \mathit{not}\ L$ iff $I(L) \preceq \mathbf{f}$,
- $I \models \mathit{not}\ \neg L$ iff $I(L) \preceq \mathbf{t}$.

The first condition indicates that if L is false in I, its default negation *not* L holds in I; else if L is undefined in I, *not* L holds in I as negation as failure to prove; otherwise $I \not\models \mathit{not}\ L$. The second condition gives the counterpart statement.

The notion of (p-minimal) models is defined in the same way as in the previous section.

The *paraconsistent stable model semantics* of an extended disjunctive program is defined as follows.

Definition 6.2 Let P be an extended disjunctive program and I be a subset of $\mathcal{L}_P$. The *reduct* of P with respect to I is the positive extended disjunctive program P^I such that a clause

$$L_1 \vee \ldots \vee L_l \leftarrow L_{l+1} \wedge \ldots \wedge L_m \tag{6.1}$$

is in P^I iff there is a ground clause of the form:

$$L_1 \vee \ldots \vee L_l \leftarrow L_{l+1} \wedge \ldots \wedge L_m \wedge not\, L_{m+1} \wedge \ldots \wedge not\, L_n \ \ (l \geq 0) \tag{6.2}$$

from P such that $\{L_{m+1}, \ldots, L_n\} \cap I = \emptyset$. Then I is called a *paraconsistent stable model* (shortly, *p-stable model*) of P if I is a p-minimal model of P^I. □

Example 6.3 Let P be the program:

$$\{\ a \vee b \leftarrow, \quad \neg a \leftarrow, \quad \neg b \leftarrow, \quad c \leftarrow not\, d\ \}.$$

Then P has two p-stable models $\{a, \neg a, \neg b, c\}$ and $\{b, \neg a, \neg b, c\}$. □

A p-stable model I is *consistent* if $I(L) \neq \top$ for any $L \in \mathcal{L}_P$, otherwise I is *inconsistent*. There is a program which has no p-stable model.

Example 6.4 The program

$$P = \{\ a \leftarrow not\, a, \quad b \leftarrow\ \}$$

has no p-stable model. □

A program which has at least one p-stable model is called *coherent*, while a program having no p-stable model is called *incoherent*. By definition, the notion of p-stable models reduces to that of p-minimal models in positive extended disjunctive programs. In extended disjunctive programs, every p-stable model is minimal.

Proposition 6.7 A p-stable model is a p-minimal model.

Proof: Let I be a p-stable model of a program P. Assume that there is a p-minimal model J of P such that $J \preceq I$. Since J is a model of P and satisfies each clause (6.2) in P, and $\{L_{m+1}, \ldots, L_n\} \cap I = \emptyset$ and $J \preceq I$ imply $\{L_{m+1}, \ldots, L_n\} \cap J = \emptyset$, J also satisfies each clause (6.1) in P^I. Thus J is a model of P^I, and since I is a p-minimal model of P^I, $J \preceq I$ implies $J = I$. □

The converse of the above proposition does not hold in general. For instance, in Example 6.4, $\{a, b\}$ is the p-minimal model of P, but not p-stable.

6.3.2 Paraconsistent Possible Model Semantics

The paraconsistent possible model semantics for positive extended disjunctive programs is also directly extended to extended disjunctive programs.

Given an extended disjunctive program P, a *split program* is defined as a ground extended logic program obtained from P by replacing each ground disjunctive clause $C: L_1 \vee \ldots \vee L_l \leftarrow L_{l+1} \wedge \ldots \wedge L_m \wedge not\, L_{m+1} \wedge \ldots \wedge not\, L_n$ with the following ground extended clauses (called *split clauses*):

$$L_i \leftarrow L_{l+1} \wedge \ldots \wedge L_m \wedge not\, L_{m+1} \wedge \ldots \wedge not\, L_n \ \text{ for every } L_i \in S$$

where S is some non-empty subset of $\{L_1, \ldots, L_l\}$. Then a *p-possible model* of P is defined as a p-stable model of any split program of P.

It is easy to see that any p-possible model is a model of P. The following properties are direct extensions of those presented in Section 3.3.

Proposition 6.8 A coherent extended disjunctive program has at least one p-possible model. □

Proposition 6.9 Any p-stable model is a minimal p-possible model, but not vice versa. □

As is the case of normal disjunctive programs, there are incoherent programs having p-possible models.

Example 6.5 Let P be the program:

$$\{\ a \vee \neg b \leftarrow, \quad \neg b \leftarrow a, \quad \leftarrow not\, a\ \}.$$

Then P has the p-possible model $\{a, \neg b\}$, while it has no p-stable model. □

6.3.3 Connection to the Answer Set Semantics

For extended disjunctive programs, Gelfond and Lifschitz [1991] have introduced the *answer set semantics*. The answer sets are defined in the same manner as p-stable models in Definition 6.2 except that the definition of p-minimal models of a positive extended disjunctive program P^I is changed in a way that $I = \mathcal{L}_P$ if a model I contains a pair of complementary literals L and $\neg L$. For instance, in Example 6.3, P has two inconsistent p-stable models, while it has the unique answer set $\mathcal{L}_P$. Thus p-stable models are paraconsistent, while answer sets are not.

In this section, we provide a connection between the p-stable model semantics and the answer set semantics in extended disjunctive programs. As presented above, the essential difference between the two semantics is the treatment of inconsistency. Then we relate p-stable models to answer sets by trivializing inconsistent p-stable models.

Let us consider a program P_{tr} obtained from P by incorporating the *trivialization rule*:

$$N \leftarrow L \wedge \neg L \tag{6.3}$$

for all literals L and N from $\mathcal{L}_P$. Then the relationship between answer sets and p-stable models of extended disjunctive programs is as follows.

Theorem 6.10 Let P be an extended disjunctive program. Then I is an answer set of P iff I is a p-stable model of P_{tr}.

Proof: Since consistent answer sets coincide with consistent p-stable models, the result follows when I is a consistent answer set. Otherwise, suppose the case that P has the contradictory answer set $\mathcal{L}_P$. Then, by the definition of answer sets, the positive extended disjunctive program $P^{\mathcal{L}_P}$ has the answer set $\mathcal{L}_P$. In this case, $P^{\mathcal{L}_P}$ has no consistent p-minimal model, but an inconsistent p-minimal model. Thus, in the presence of the trivialization rule (6.3), $P^{\mathcal{L}_P}_{tr}$ has the p-minimal model containing every literal N from $\mathcal{L}_P$. Hence, $\mathcal{L}_P$ is the unique p-stable model of P_{tr}. On the other hand, if I is an inconsistent p-stable model of P_{tr}, it contains every literal N from $\mathcal{L}_P$ by (6.3). In this case, $\mathcal{L}_P$ is the p-minimal model of $P^{\mathcal{L}_P}_{tr}$, and thus each p-minimal model of $P^{\mathcal{L}_P}$ contains a pair of complementary literals. Hence, $\mathcal{L}_P$ is the unique answer set of P. □

The above theorem indicates that we can easily simulate the "classical" meaning of logic programming by a simple program transformation. Note that without the trivialization rule, there is no one-to-one correspondence between inconsistent p-stable models and the answer set $\mathcal{L}_P$ in general.

Example 6.6 The program

$$\{\ \neg a \leftarrow,\quad a \leftarrow not\ b\ \}$$

has no answer set, while it has an inconsistent p-stable model $\{\neg a,\ a\}$. On the other hand, the program

$$\{\ a \leftarrow,\quad \neg a \leftarrow,\quad b \leftarrow not\ b\ \}$$

has the answer set $\mathcal{L}_P$, while it has no p-stable model. □

The example illustrates that a program possibly has an inconsistent p-stable model even when there is no answer set of the program. When a program has no p-stable model, on the other hand, the program has either no answer set or the trivial answer set $\mathcal{L}_P$. Since the answer set semantics brings no useful information, the absence of p-stable models in this case is not a serious drawback. From this observation, the paraconsistent stable model semantics is considered to be more useful than the answer set semantics.

By Proposition 6.9, the answer set semantics is related to the paraconsistent possible model semantics as follows.

Corollary 6.11 Let P be an extended disjunctive program. If I is an answer set of P, then I is a minimal p-possible model of P_{tr}. □

As previously discussed, paraconsistent possible models are well-defined whenever paraconsistent stable models are. Then, together with the above observation, we can conclude that the paraconsistent possible model semantics is the best choice among others as a semantics of extended disjunctive programs.

P-stable models of an extended disjunctive program are also characterized by stable models of the *positive form* of the program. Recall that a positive form of an extended disjunctive program P is defined as a normal disjunctive program P^+ which is obtained by replacing each negative literal $\neg L$ in P with a corresponding newly introduced atom L' in P^+. Let I^+ be a model of such P^+. Then the following relation holds by definition.

Proposition 6.12 Let P be an extended disjunctive program and P^+ be its positive form. Then I is a p-stable model of P iff I^+ is a stable model of P^+. □

Note that in case of the answer set semantics the above relation holds only for consistent answer sets [Gelfond and Lifschitz, 1991].

6.4 Fixpoint Semantics of Extended Disjunctive Programs

A fixpoint semantics of positive extended disjunctive programs is defined in the same manner as that of positive disjunctive programs in Section 3.4 except that in this case the closure operator acts over the lattice of sets of interpretations $2^{2^{\mathcal{L}_P}}$.

Given a positive extended disjunctive program P, a mapping $T_P : 2^{2^{\mathcal{L}_P}} \rightarrow 2^{2^{\mathcal{L}_P}}$ is defined as in Definition 3.7, and its fixpoint closure $T_P \uparrow \omega$ is defined as well.

Let $\mathcal{PMM}_P$ be the set of all p-minimal models, and $\mathcal{PPM}_P$ be the set of all p-possible models of a program P. Then the following results hold.

Theorem 6.13 Let P be a positive extended disjunctive program. Then,

(i) $\mathcal{PPM}_P = \mu(T_P \uparrow \omega)$.

(ii) $\mathcal{PMM}_P = min(\mu(T_P \uparrow \omega))$.

Proof: Viewing each negative literal in P as an atom, the results directly follow from Theorem 3.17 in Section 3.4. □

Corollary 6.14 Let P be a positive extended logic program. Then $T_P \uparrow \omega$ contains the unique p-least model of P. □

The above corollary corresponds to Blair and Subrabmanian's fixpoint semantics of paraconsistent logic programs [Blair and Subrahmanian, 1989].

The fixpoint semantics is also generalized to extended disjunctive programs. Given an extended disjunctive program P, let us define the notions of the epistemic transformation P^κ, canonical interpretations, and the function $obj_c(\mathcal{I}_{P^\kappa})$ in the same way as those of normal disjunctive programs. Let $\mathcal{PST}_P$ be the set of all p-stable models of an extended disjunctive program P. Then the following results hold.

Theorem 6.15 Let P be an extended disjunctive program. Then,

(i) $\mathcal{PPM}_P = obj_c(\mu(T_{P^\kappa} \uparrow \omega))$.

(ii) $\mathcal{PST}_P = obj_c(min(\mu(T_{P^\kappa} \uparrow \omega)))$.

In particular, when P is an extended logic program, $\mathcal{PST}_P = obj_c(\mu(T_{P^\kappa} \uparrow \omega))$.

Proof: The results follow from Lemma 3.20, Lemma 3.21, and Theorem 3.22 in Section 3.4. □

As for the answer set semantics, the following relation holds by Theorem 6.10.

Corollary 6.16 Let P be an extended disjunctive program and $\mathcal{AS}_P$ be the set of all answer sets of P. Then,

$$\mathcal{AS}_P = obj_c(min(\mu(T_{P^\kappa_{tr}} \uparrow \omega)))$$

where P^κ_{tr} is the epistemic transformation of P_{tr}. In particular, when P is an extended logic program, $\mathcal{AS}_P = obj_c(\mu(T_{P^\kappa_{tr}} \uparrow \omega))$. □

6.5 Reasoning with Inconsistency

This section presents applications of paraconsistent semantics for reasoning with inconsistent information. In this section, we discuss the issue based on the paraconsistent stable model semantics, while the techniques are directly applicable to the paraconsistent possible model semantics.

6.5.1 Preferred Stable Models

In the previous section, we have defined the paraconsistent stable model semantics by the collection of all p-stable models. However, when a program has consistent models as well as inconsistent ones, a rational reasoner may prefer consistent models to inconsistent ones and consider truth values only in consistent models.

Example 6.7 Let P be the program:

$$\{\ \neg a \vee b \leftarrow,\quad a \leftarrow \neg c,\quad \neg c \leftarrow not\ c\ \},$$

which has two p-stable models $\{a, \neg a, \neg c\}$ and $\{a, b, \neg c\}$. In this case, however, it seems natural to prefer the consistent model $\{a, b, \neg c\}$ and conclude the truth of a and b. □

When an extended disjunetive program has consistent p-stable models, we distinguish these consistent p-stable models as *preferred p-stable models*. Thus preferred p-stable models characterize "consistent" meaning of a program. In fact, preferred p-stable models coincide with consistent answer sets of extended disjunctive programs.

Theorem 6.17 Let P be an extended disjunctive program. Then I is a preferred p-stable model of P iff I is a consistent answer set of P.

Proof: Since consistent answer sets coincide with consistent p-stable models, the result follows. □

6.5.2 Suspicious Stable Models

When a program contains inconsistent information, it is useful to distinguish facts affected by such information from other information in a program.

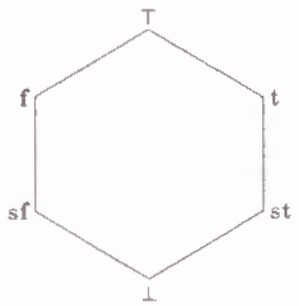


Figure 6.2: Six-valued lattice VI

Example 6.8 Let P be the program:

$$\{\, a \leftarrow b \wedge not\, c, \quad b \leftarrow, \quad \neg b \leftarrow, \quad d \leftarrow \,\}.$$

Then P has the p-stable model $\{a, b, \neg b, d\}$. However, the truth of a is less credible than the truth of d, since a is derived through the contradictory fact b. □

In order to distinguish such suspicious facts from others, we present suspicious reasoning under the paraconsistent stable model semantics. To this end, we first introduce two new truth values **st** and **sf**, which respectively denote *suspiciously true* and *suspiciously false*. These newly introduced values together with the values in IV constitute a lattice of six-valued logic VI such that $\bot \preceq \mathbf{sx} \preceq \mathbf{x} \preceq \top$ for $\mathbf{x} \in \{\mathbf{t}, \mathbf{f}\}$ (Figure 6.2).

Let $\mathcal{L}_P^s = \mathcal{L}_P \cup \{L^s \mid L \in \mathcal{L}_P\}$ and I^s be a subset of $\mathcal{L}_P^s$, where each adorned literal L^s denotes a *suspicious literal*. Then an interpretation under the logic VI is defined as a function $I^s : \mathcal{L}_P^s \to VI$ such that for each literal $L \in \mathcal{L}_P$,

$$\begin{aligned} I^s(L) \;=\; lub\{x \mid\; & x = \mathbf{t} \quad \text{if } L \in I^s, \\ & x = \mathbf{f} \quad \text{if } \neg L \in I^s, \\ & x = \mathbf{st} \quad \text{if } L^s \in I^s, \\ & x = \mathbf{sf} \quad \text{if } \neg L^s \in I^s, \\ & x = \bot \quad \text{otherwise}\,\}. \end{aligned}$$

That is, the truth value of each literal $I^s(L)$ is defined as the least upper bound of each value x which is determined by the literal occurrences in I^s. Thus, $I^s(L) = \top$ iff either $L \in I^s$ and $\neg L \in I^s$, or $L^s \in I^s$ and $\neg L^s \in I^s$, or $L \in I^s$ and $\neg L^s \in I^s$, or

$\neg L \in I^s$ and $L^s \in I^s$. Note that $I^s(L) = \mathbf{st}$ iff $I^s(\neg L) = \mathbf{sf}$. Under the logic VI, satisfaction of literals and default negation is defined as follows.

- $I^s \models L$ iff $\mathbf{st} \preceq I^s(L)$,
- $I^s \models \neg L$ iff $\mathbf{sf} \preceq I^s(L)$,
- $I^s \models not\, L$ iff $I^s(L) \preceq \mathbf{f}$,
- $I^s \models not\, \neg L$ iff $I^s(L) \preceq \mathbf{t}$.

Satisfaction of clauses is the same as before.

Next, for a positive extended disjunctive program P and an interpretation I^s, let $\mathcal{T}_P^s$ be a mapping which is defined in the same way as in Definition 3.7 except that we consider the mapping T_P^s instead of T_P as follows: if $I^s \models L_1 \wedge \ldots \wedge L_m$ for some ground integrity constraint $\leftarrow L_1 \wedge \ldots \wedge L_m$ from P, then $T_P^s(I^s) = \emptyset$; otherwise,

$$\begin{aligned} T_P^s(I^s) \;=\; \{\, J^s \mid\; & \text{for each ground clause } C_i :\; L_1 \vee \ldots \vee L_{l_i} \leftarrow L_{l_i+1} \wedge \ldots \wedge L_{m_i}, \\ & \text{from } P \text{ such that } I^s \models L_{l_i+1} \wedge \ldots \wedge L_{m_i}, \\ & J^s = I^s \cup \bigcup_{C_i} \{L_j'\}\ (1 \le j \le l_i) \quad \text{where} \\ & L_j' = L_j \quad \text{if } L_k \in I^s \text{ and } I^s \not\models \neg L_k \text{ for each } L_k\ (l_i + 1 \le k \le m_i); \\ & L_j' = L_j^s, \quad \text{otherwise}\,\}. \end{aligned}$$

The intuitive meaning of T_P^s is that when the body of a clause C_i is satisfied by I^s, each derived disjunct $L_j' = L_j$ is added to I^s if any literal L_k in the body is derived without suspicion and its negative counterpart $\neg L_k$ or $\neg L_k^s$ is not included in I^s. Otherwise, the derived disjunct is suspicious $L_j' = L_j^s$, since it is derived through inconsistent information in a program.

Given an extended disjunctive program P and its epistemic transformation P^κ, let us consider the fixpoint closure $\mathcal{SPST}_P = obj_c(min(\mu(\mathcal{T}_{P^\kappa}^s \uparrow \omega)))$. We call $\mathcal{SPST}_P$ the *suspicious p-stable models* of P.

Theorem 6.18 Let P be an extended disjunctive program. Then a suspicious p-stable model is a model of P.

Proof: In a suspicious p-stable model, the truth value of each literal possibly becomes **st** or **sf** when its truth value is respectively **t** or **f** in its corresponding p-stable model. Let I^s be a suspicious p-stable model and I be its corresponding p-stable model in which each literal L^s is identified

with L. By definition, $I^s \models L$ iff $I \models L$, and $I^s \models not\ L$ iff $I \models not\ L$. Thus I^s satisfies each clause of P whenever I is a p-stable model of P. Hence the result follows. □

Corollary 6.19 For each suspicious p-stable model I^s of P, $I = \{L \mid I^s \models L\}$ is a p-stable model of P. Conversely, for each p-stable model J of P, there is a suspicious p-stable model J^s of P such that $J = \{L \mid J^s \models L\}$. □

Thus suspicious p-stable models can distinguish information derived through contradictory facts from other information. Clearly, suspicious p-stable models reduce to p-stable models in the absence of suspicious information.

Note that a proved fact is considered to be suspicious if every proof of the fact includes inconsistent information. Then if an interpretation includes both L and L^s, it means that there is a proof of L depending on no inconsistent information. In this case, by taking the least upper bound of **t** and **st**, the truth value of L becomes **t**.

Example 6.9 (cont. from Example 6.8)
P^κ becomes

$$\{\ \lambda \vee \mathsf{K}c \leftarrow b,\quad a \leftarrow \lambda,\quad \leftarrow \lambda \wedge c,\quad b \leftarrow,\quad \neg b \leftarrow,\quad d \leftarrow\ \},$$

and

$$obj_c(min(\mu(\mathcal{T}^s_{P^\kappa} \uparrow \omega))) = \{\{a^s,\ b,\ \neg b,\ d\}\}.$$

Then, $I^s(a) = \mathbf{st}$, $I^s(b) = \top$, $I^s(c) = \bot$, and $I^s(d) = \mathbf{t}$. □

6.5.3 Semi-Stable Models

There is an extended disjunctive program which has no p-stable model but still contains useful information. For instance, in Example 6.4, P has no p-stable model but it seems reasonable to conclude the truth of b. Roughly speaking, incoherency arises when a literal is implied by its default negation in a program. Since incoherency is viewed as a kind of inconsistency, it is desirable to provide a framework which is paraconsistent for such incoherency. In this section, we introduce the notion of semi-stable models which is paraconsistent for incoherent programs.

To represent incoherent facts, we first introduce five extra truth values **bt**, **bf**, **b⊤**, **tcb**, and **fcb** which respectively denote *believed true*, *believed false*, *believed contradictory*, *true with contradictory belief* and *false with contradictory belief*. These values together with the values in IV constitute a lattice of *nine-valued logic* IX such that $\bot \preceq \mathbf{bx} \preceq \mathbf{x} \preceq \mathbf{xcb} \preceq \top$ and $\mathbf{bx} \preceq \mathbf{b}\top \preceq \mathbf{xcb}$ for $\mathbf{x} \in \{\mathbf{t}, \mathbf{f}\}$ (Figure 6.3).

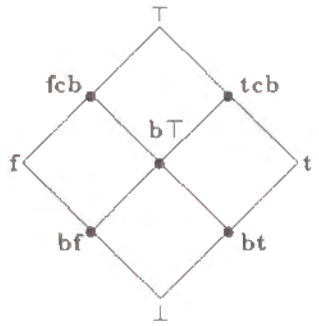


Figure 6.3: Nine-valued lattice IX

Let $\mathcal{L}^\kappa_P = \mathcal{L}_P \cup \{\mathsf{K}L \mid L \in \mathcal{L}_P\}$ and I^κ be a subset of $\mathcal{L}^\kappa_P$. Then an interpretation under the logic IX is defined as a function $I^\kappa : \mathcal{L}^\kappa_P \to IX$ such that for each literal $L \in \mathcal{L}_P$,

$$\begin{aligned} I^\kappa(L) \;=\; lub\{x \mid\ & x = \mathbf{t} \quad \text{if } L \in I^\kappa, \\ & x = \mathbf{f} \quad \text{if } \neg L \in I^\kappa, \\ & x = \mathbf{bt} \quad \text{if } \mathsf{K}L \in I^\kappa, \\ & x = \mathbf{bf} \quad \text{if } \mathsf{K}\neg L \in I^\kappa, \\ & x = \bot \quad \text{otherwise}\ \}. \end{aligned}$$

Thus, $I^\kappa(L) = \mathbf{b}\top$ iff both $\mathsf{K}L \in I^\kappa$ and $\mathsf{K}\neg L \in I^\kappa$; $I^\kappa(L) = \mathbf{fcb}$ iff both $\mathsf{K}L \in I^\kappa$ and $\neg L \in I^\kappa$; $I^\kappa(L) = \mathbf{tcb}$ iff both $\mathsf{K}\neg L \in I^\kappa$ and $L \in I^\kappa$, and so on. Note that $I^\kappa(L) = \mathbf{bt}$ iff $I^\kappa(\neg L) = \mathbf{bf}$, $I^\kappa(L) = \mathbf{b}\top$ iff $I^\kappa(\neg L) = \mathbf{b}\top$, and $I^\kappa(L) = \mathbf{tcb}$ iff $I^\kappa(\neg L) = \mathbf{fcb}$.

The intuitive reading of each newly introduced truth value is that if $I^\kappa(L) = \mathbf{bt}$, I^κ contains a belief $\mathsf{K}L$ without its justification L. On the other hand, if $I^\kappa(L) = \mathbf{tcb}$, I^κ contains a fact L with its opposite belief $\mathsf{K}\neg L$.

Under this logic, satisfaction of literals and default negation is defined in the same way as Section 6.3, i.e., $I \models L$ iff $\mathbf{t} \preceq I(L)$; $I \models \neg L$ iff $\mathbf{f} \preceq I(L)$; $I \models not\ L$ iff $I(L) \preceq \mathbf{f}$; and $I \models not\ \neg L$ iff $I(L) \preceq \mathbf{t}$. Satisfaction of clauses is also defined as before.

According to the above definition, when $I(L) = \mathbf{bt}$ or $I(L) = \mathbf{b}\top$, it holds that $I \not\models L$ and $I \not\models not\ L$. This means when L is believed true, it is too weak to conclude the truth of L, but enough to reject $not\ L$.[3] Else when $I(L) = \mathbf{tcb}$, $I \models L$ while

[3]Recall that *not* L corresponds to $\neg\mathsf{K}L$.

$I \not\models not \neg L$. This means when L is true with contradictory belief, I concludes the truth of L but rejects $not\ \neg L$ in the presence of its opposite belief $\mathsf{K}\neg L$.

Next let $\mathcal{I}_{P^\kappa}$ be a set of interpretations of a program P^κ obtained by the epistemic transformation of an extended disjunctive program P. Then an interpretation $I^\kappa \in \mathcal{I}_{P^\kappa}$ is said *maximally canonical* if there is no interpretation $J^\kappa \in \mathcal{I}_{P^\kappa}$ such that $\{\mathsf{K}L \mid \mathsf{K}L \in J^\kappa \text{ and } L \notin J^\kappa\} \subset \{\mathsf{K}L \mid \mathsf{K}L \in I^\kappa \text{ and } L \notin I^\kappa\}$. That is, a maximally canonical interpretation is an interpretation such that the canonical condition is satisfied as much as possible. In particular, if $\mathcal{I}_{P^\kappa}$ contains an interpretation I^κ which is canonical, it is also maximally canonical. Now let

$$obj^\kappa_{mc}(\mathcal{I}_{P^\kappa}) = \{I^\kappa \cap \mathcal{L}^\kappa_P \mid I^\kappa \in \mathcal{I}_{P^\kappa} \text{ and } I^\kappa \text{ is maximally canonical }\}.$$

Theorem 6.20 Let P be an extended disjunctive program. Then any interpretation included in $\mathcal{SST}_P = obj^\kappa_{mc}(min(\mu(\mathcal{T}_{P^\kappa} \uparrow \omega)))$ is a model of P.

Proof: By definition, each maximally canonical interpretation I^κ included in $min(\mu(\mathcal{T}_{P^\kappa} \uparrow \omega))$ is a model of P^κ. Then, for each transformed clauses:

$$\lambda_1 \vee \ldots \vee \lambda_l \vee \mathsf{K}L_{m+1} \vee \ldots \vee \mathsf{K}L_n \leftarrow L_{l+1} \wedge \ldots \wedge L_m\,,$$
$$L_i \leftarrow \lambda_i \quad \text{for } i = 1, \ldots, l,$$

$\{L_{l+1}, \ldots, L_m\} \subseteq I^\kappa$ implies either $L_i \in I^\kappa$ $(1 \leq i \leq l)$ or $\mathsf{K}L_j \in I^\kappa$ $(m+1 \leq j \leq n)$. In case of $L_i \in I^\kappa$, I^κ satisfies the corresponding clause:

$$L_1 \vee \ldots \vee L_l \leftarrow L_{l+1} \wedge \ldots \wedge L_m \wedge not\ L_{m+1} \wedge \ldots \wedge not\ L_n \quad (*)$$

in P. In case of $\mathsf{K}L_j \in I^\kappa$, when $L_j \in I^\kappa$, I^κ satisfies the clause $(*)$ in P. Else when $L_j \notin I^\kappa$, (i) if $\neg L_j \notin I^\kappa$, the truth value of L_j is $\mathbf{bt}$ or $\mathbf{b}\top$, then $I^\kappa \not\models not\ L_j$. (ii) Else if $\neg L_j \in I^\kappa$, the truth value of L_j becomes $\mathbf{fcb}$, then $I^\kappa \not\models not\ L_j$. In either case, I^κ satisfies the clause $(*)$ in P. Therefore, I^κ satisfies each clause in P. Hence, $I^\kappa \cap \mathcal{L}^\kappa_P$, which is obtained from I^κ by removing every λ_i, is also a model of P. □

We call models $\mathcal{SST}_P$ the *semi-stable models* of P.
The notion of semi-stable models reduces to p-stable models in coherent programs.

Corollary 6.21 Let P be a coherent program. Then its semi-stable models coincide with the p-stable models.

Proof: When $min(\mu(\mathcal{T}_{P^\kappa} \uparrow \omega))$ contains canonical interpretations, they are also maximally canonical. Hence the result follows by definition. □

The existence of semi-stable models is guaranteed for any program which has models.

Theorem 6.22 When a program has a model, it has a semi-stable model.

Proof: When a program P has a model, it is easy to see that P^κ also has a model. Then the closure $min(\mu(\mathcal{T}_{P^\kappa} \uparrow \omega))$ contains models which are maximally canonical, hence $\mathcal{SST}_P$ is not empty. □

Thus incoherent programs get the meaning by considering semi-stable models.

Example 6.10 (Barbar's Paradox)
Let P be the program:

$$\{\ shave(Noel, x) \leftarrow not\ shave(x, x), \quad mayor(Casanova) \leftarrow\ \}.$$

Then its epistemic transformation P^κ becomes

$$\begin{aligned}&\{\ \lambda(x) \vee \mathsf{K}shave(x, x) \leftarrow,\\ &shave(Noel, x) \leftarrow \lambda(x),\\ &\leftarrow \lambda(x) \wedge shave(x, x),\\ &mayor(Casanova) \leftarrow\ \}.\end{aligned}$$

Thus,

$$\begin{aligned}min(\mu(\mathcal{T}_{P^\kappa} \uparrow \omega)) \;=\; &\{\{\mathsf{K}shave(N, N),\ \mathsf{K}shave(C, C),\ mayor(C)\},\\ &\{\mathsf{K}shave(N, N),\ \lambda(C),\ shave(N, C),\ mayor(C)\}\}.\end{aligned}$$

In the above closure, the second interpretation is maximally canonical, hence

$$obj^\kappa_{mc}(min(\mu(\mathcal{T}_{P^\kappa} \uparrow \omega))) = \{\{\ \mathsf{K}shave(N, N),\ shave(N, C),\ mayor(C)\ \}\},$$

which contains the unique semi-stable model of P such that $shave(N, C)$ and $mayor(C)$ are true, while $shave(N, N)$ is believed true. □

Note that the above program has neither standard two-valued stable models nor answer sets.

In the incoherent program $\{\ a \leftarrow not\ a\ \}$, it is known that interpretations "oscillate" between $\emptyset$ and $\{a\}$ under the *stable class semantics* [Baral and Subrahmanian, 1992]. Then it is interesting to observe that the truth value $I(a) = \mathbf{bt}$ in its semi-stable model correspondingly lies between $\bot$ and $\mathbf{t}$.

6.6 Related Work

A framework of paraconsistent logic programming is firstly developed by Blair and Subrahmanian [1989] in the context of annotated logic programs. They employ Belnap's four-valued logic as a theoretical basis, but their framework does not treat default negation in a program. Fitting [1991] provides a general framework for logic programming in terms of bilattices, but he does not discuss programs containing two kinds of negation. Kifer and Lozinskii [1992] extend Blair and Subrahmanian's annotated logic programming framework to a theory possibly containing default negation, and Wagner [1991a; 1991b] also develops a theory of inconsistent logic programs with two kinds of negation. Compared with our approach, they do not treat disjunctions in a program and the underlying logics presented in these literature are different from our stable model semantics.

Subrahmanian [1992] has extended the work of [Blair and Subrahmanian, 1989] to programs containing disjunctive information. However, he does not treat default negation in a program. He also provides a fixpoint semantics of paraconsistent disjunctive programs based on Minker and Rajasekar's model state fixpoint semantics, which is different from ours as presented in Section 3.7. Lu and Henschen [1992] consider specifying the closed world assumption in paraconsistent definite and disjunctive logic programs. However, they neither consider programs containing two kinds of negation nor develop any fixpoint theory for disjunctive programs.

Paraconsistent stable model semantics is also proposed by several researchers. Pimentel and Rodi [1991], Grant and Subrahmanian [1992], and Wagner [1993] study paraconsistent stable model semantics from different viewpoints. The differences between these approaches and ours are as follows. First, their paraconsistent stable model semantics are defined for extended logic programs and do not treat disjunctive information in a program. Second, they do not provide any mechanism to compute their stable models, while our fixpoint computation realizes constructive computation of paraconsistent stable models by using bottom-up model generation techniques as presented in Section 3.5. Third, we have introduced the notion of semi-stable models which are paraconsistent for incoherent programs, while they do not discuss the issue of handling incoherency. Recently, Fitting [1993] provided a framework of stable model semantics in terms of bilattices, but did not treat disjunctive programs. Another point is that our paraconsistent extensions are not only for the stable model semantics, but also for the possible model semantics. Sakama [1992] has also developed a paraconsistent well-founded semantics for extended logic programs and disjunctive programs, which is different from the stable model approach presented in this chapter.

The paraconsistent semantics presented in this chapter is intended to localize inconsistent information in a program. There is an alternative approach which tries to detect the source of inconsistency and recover the consistency of a program. Generally speaking, however, it is a hard task to automatically resolve inconsistency in a program. When inconsistency arises from default assumptions, Pereira et al. [1991] and Dung et al. [1991] present methods of removing the inconsistency by preferring a fact that does not depend on any default assumption. However, their approaches are of no use in a program where inconsistency is derived without default assumptions. Kowalski and Sadri [1990] resolve contradiction by giving a higher priority to one of the conflicting conclusions as an exception, but such an approach generally requires one to specify a preference for each individual rule. Inoue [1991] and Baral et al. [1992b] consider the meaning of an inconsistent program as a collection of maximally consistent subsets of the program, but such a collection exponentially grows according to the increase of inconsistent information.

6.7 Summary

This chapter has presented paraconsistent frameworks for extended disjunctive programs. We have introduced the paraconsistent minimal, stable, and possible model semantics for extended disjunctive programs based on lattice-structured multi-valued logics. The paraconsistent semantics are characterized by a fixpoint semantics of extended disjunctive programs. We have also discussed applications of the paraconsistent semantics for reasoning with inconsistency.

The paraconsistent semantics are natural extensions of those corresponding semantics for normal disjunctive programs, and compared with Gelfond and Lifschitz's answer set semantics, the proposed semantics do not trivialize a program in the presence of inconsistent information. The paraconsistent semantics presented in this chapter generalize previous studies of paraconsistent logic programming and provide a uniform framework of logic programming possibly containing inconsistent information, disjunctive information, integrity constraints, and both explicit and default negation in a program. From the computational viewpoint, the bottom-up model generation procedure presented in Section 3.5 is used to compute the p-stable/p-possible models, and complexity results presented in Section 3.6 are directly applicable to the corresponding paraconsistent semantics. Thus the paraconsistent possible model semantics has a computational advantage over the paraconsistent minimal and stable model semantics, and the answer set semantics.

Chapter 7

Partial Deduction of Disjunctive Logic Programs

In this chapter, we present partial deduction of disjunctive programs. We first show that normal partial deduction in logic programming is not applicable as it is in the context of disjunctive programs. Then we introduce a new partial deduction technique for disjunctive programs, and show that it preserves the minimal model semantics of positive disjunctive programs, and the disjunctive stable model semantics of normal disjunctive programs. Normal partial deduction is also used together with suitable program transformations from disjunctive programs to normal logic programs, and the possible model semantics is preserved through such transformations. Partial deduction techniques are also applied to goal-oriented partial deduction for query optimization.

7.1 Introduction

Logic programming provides a methodology as a declarative programming language, while a correctly specified logic program is not necessarily efficient as a practical programming language. In order to bridge the gap between declarative and practical programming, studies have been devoted to develop techniques for optimizing logic programs.

Partial deduction or *partial evaluation* is known as one of the optimization techniques in logic programming. Given a logic program, partial deduction derives a more specific program through performing deduction on a part of the program, while preserving the meaning of the original program. Such a specialized program is usually more efficient than the original program when executed.

Partial deduction in logic programming was firstly introduced by Komorowski [1981] and has been developed by several researchers from various viewpoints [Sestoft and Zamulin, 1988; Komorowski, 1992]. From the semantic point of view, Lloyd and Shepherdson [1991] formalized partial evaluation for normal logic programs and provide the conditions to assure the correctness with respect to Clark's program completion semantics. On the other hand, Tamaki and Sato [1984] showed that partial deduction preserves the least Herbrand model semantics of definite logic programs in the context of *unfold/fold* transformation. The result is extended to the perfect model semantics for stratified logic programs [Seki, 1991; Maher, 1993], and the well-founded semantics for normal logic programs [Seki, 1993].

When we consider disjunctive programs, they increase expressive power of logic programming on the one hand, but their computation is generally expensive on the other hand. Then optimizations of disjunctive programs are important issues for practical usage, however, the partial deduction technique in normal logic programs is not applicable to disjunctive programs in its present form. This is because normal partial deduction is based on unfolding between normal clauses, and it supplies no mechanism for unfolding between disjunctive clauses.

In this chapter, we develop partial deduction techniques for disjunctive programs. We first show that normal partial deduction is not useful in the presence of disjunctive information in a program, then introduce disjunctive partial deduction for disjunctive programs. We prove that the proposed partial deduction method preserves the minimal model semantics of positive disjunctive programs, and the disjunctive stable model semantics of normal disjunctive programs. We also discuss the preservation of the possible model semantics, and present goal-oriented partial deduction for query optimization.

The rest of this chapter is organized as follows. In Section 7.2, we present disjunctive partial deduction for positive disjunctive programs and show its correctness with respect to the minimal model semantics. Section 7.3 extends the result to normal disjunctive programs containing default negation, and shows that proposed partial deduction also works well for the disjunctive stable model semantics. A connection between normal and disjunctive partial deduction is presented, and the preservation of the possible model semantics is discussed. In Section 7.4, the partial deduction technique is applied to goal-oriented partial deduction for query optimization. Section 7.5 discusses further issues, and Section 7.6 summarizes this chapter.

7.2 Partial Deduction of Positive Disjunctive Programs

In this section, we first present partial deduction of positive disjunctive programs. In the following, when we write $A \vee \Sigma \leftarrow \Gamma$, Σ denotes a disjunction (possibly *false*) in the head, and Γ denotes a conjunction (possibly *true*) in the body. Note here that when we write a clause as $A \vee \Sigma \leftarrow \Gamma$, it does not necessarily mean that A should be the leftmost atom in the head of the clause. That is, any two clauses are identified modulo the permutation of disjuncts/conjuncts in their heads/bodies.

Since a program is semantically identified with its ground program, we consider ground programs throughout this chapter unless stated otherwise. We also assume without loss of generality that a disjunction in the head of a ground clause is already *factored*, that is, each atom in the disjunctive head of a clause is different.

7.2.1 Normal Partial Deduction

Partial deduction in logic programming is usually defined as *unfolding* of clauses in a program.[1] For a Horn logic program P, partial deduction is formally presented as follows.

Given a Horn clause C from P:

$$C: \ H \leftarrow A \wedge \Gamma,$$

suppose that $C_1, \ldots, C_k$ are all of the clauses in P such that each of which has the atom A in its head:

$$C_i: \ A \leftarrow \Gamma_i \ \ (1 \leq i \leq k).$$

Then *normal partial deduction* of P (with respect to C on A) is defined as the program $\pi^N_{\{C;A\}}(P)$ (called a *residual program*) such that

$$\pi^N_{\{C;A\}}(P) = (P \setminus \{C\}) \cup \{C'_1, \ldots, C'_k\}$$

where each C'_i is defined as

$$C'_i: \ H \leftarrow \Gamma \wedge \Gamma_i.$$

When we simply say normal partial deduction of P (written $\pi^N(P)$), it means normal partial deduction of P with respect to any clause on any atom.

[1] Partial deduction is also called partial evaluation. However, we prefer to use the term partial deduction, since partial evaluation often includes non-deductive procedures.

Example 7.1 Let P be the program:

$$P = \{\ a \leftarrow b, \quad b \leftarrow c, \quad b \leftarrow a, \quad c \leftarrow \}.$$

Then normal partial deduction of P with respect to $a \leftarrow b$ on b becomes

$$\pi^N_{\{a \leftarrow b;b\}}(P) = \{\ a \leftarrow c, \quad a \leftarrow a, \quad b \leftarrow c, \quad b \leftarrow a, \quad c \leftarrow \}. \quad \square$$

In the context of unfold/fold transformation of logic programs, Tamaki and Sato [1984] showed that normal partial deduction preserves the least Herbrand model semantics of definite logic programs.

Lemma 7.1 ([Tamaki and Sato, 1984]) Let P be a definite logic program and M_P be its least Herbrand model. Then, for any residual program $\pi^N(P)$ of P, $M_P = M_{\pi^N(P)}$. $\square$

The result also holds for Horn logic programs, that is, programs containing integrity constraints.

Theorem 7.2 Let P be a Horn logic program and $\pi^N(P)$ be any residual program of P. Then $M_P = M_{\pi^N(P)}$.

Proof: By identifying each integrity constraint $\leftarrow G$ with $false \leftarrow G$, M_P contains $false$ iff $M_{\pi^N(P)}$ contains $false$. In this case, both programs are inconsistent. Then the result follows from Lemma 7.1. $\square$

Thus, in what follows we do not take special care for the treatment of integrity constraints, that is, they are identified with normal clauses during partial deduction as presented above.

Now we consider partial deduction in disjunctive programs. If we consider to extend normal partial deduction to a program possibly containing disjunctive clauses, however, normal partial deduction does not preserve the minimal models of the program.

Example 7.2 Let P be the program:

$$P = \{\ a \vee b \leftarrow, \quad a \leftarrow d, \quad c \leftarrow a\ \}$$

where the set of all minimal models of P becomes $\mathcal{MM}_P = \{\{a, c\}, \{b\}\}$. On the other hand,

$$\pi^N_{\{c \leftarrow a;a\}}(P) = \{\ a \vee b \leftarrow, \quad a \leftarrow d, \quad c \leftarrow d\ \}$$

where $\mathcal{MM}_{\pi^N_{\{c \leftarrow a;a\}}(P)} = \{\{a\}, \{b\}\}$. $\square$

The problem is that normal partial deduction of logic programs is defined as unfolding between normal clauses. In the above example, however, there is the disjunctive clause $a \vee b \leftarrow$ containing the atom a in its head, so unfolding between $c \leftarrow a$ and $a \vee b \leftarrow$ would be needed.

Then our first task is to extend the normal partial deduction method to the one which supplies unfolding for disjunctive clauses.

7.2.2 Disjunctive Partial Deduction

Partial deduction of positive disjunctive programs is defined as follows.

Definition 7.1 Let P be a positive disjunctive program and C be a clause in P of the form:

$$C:\ \Sigma \leftarrow A \wedge \Gamma. \tag{7.1}$$

Suppose that $C_1, \ldots, C_k$ are all of the clauses in P such that each of which includes the atom A in its head:

$$C_i:\ A \vee \Sigma_i \leftarrow \Gamma_i, \quad (1 \leq i \leq k). \tag{7.2}$$

Then *disjunctive partial deduction* of P (with respect to C on A) is defined as the program $\pi^D_{\{C;A\}}(P)$ (called a *residual program*) such that

$$\pi^D_{\{C;A\}}(P) = (P \setminus \{C\}) \cup \{C'_1, \ldots, C'_k\}$$

where each C'_i is defined as

$$C'_i:\ \Sigma \vee \Sigma_i \leftarrow \Gamma \wedge \Gamma_i, \tag{7.3}$$

in which $\Sigma \vee \Sigma_i$ is factored. $\square$

Disjunctive partial deduction is a natural extension of normal partial deduction. In fact, the clause (7.3) is a resolvent of the clauses (7.1) and (7.2). In Horn logic programs, disjunctive partial deduction coincides with normal partial deduction.

Now we show that disjunctive partial deduction preserves the minimal model semantics of positive disjunctive programs. We first present a preliminary lemma.

Lemma 7.3 Let P be a positive disjunctive program and M be its minimal model. Then an atom A is in M iff there is a clause $C:\ A \vee \Sigma \leftarrow \Gamma$ in P such that $M \setminus \{A\} \models \Gamma$ and $M \setminus \{A\} \not\models \Sigma$.

Proof: ($\Rightarrow$) Suppose that for some atom A in M, there is no clause C in P such that $M \setminus \{A\} \models \Gamma$ and $M \setminus \{A\} \not\models \Sigma$. Then, for each clause C, $M \setminus \{A\} \not\models \Gamma$ or $M \setminus \{A\} \models \Sigma$, and hence it holds that $M \setminus \{A\} \models \Gamma$ implies $M \setminus \{A\} \models \Sigma$. In this case, since $M \setminus \{A\}$ satisfies each clause C, it becomes a model of P, which contradicts the assumption that M is a minimal model. Hence the result follows.

($\Leftarrow$) Assume that A is not in M. Then $M \setminus \{A\} = M$, and for a clause C in P, $M \models \Gamma$ and $M \not\models \Sigma$ imply $A \in M$, contradiction. □

Theorem 7.4 Let P be a positive disjunctive program and $\pi^D(P)$ be any residual program of P. Then $\mathcal{MM}_P = \mathcal{MM}_{\pi^D(P)}$.

Proof: ($\subseteq$) Let M be a minimal model of P. Since the clause (7.3) is a resolvent of the clauses (7.1) and (7.2) in P, M also satisfies each clause (7.3) in $\pi^D(P)$. Then M is a model of $\pi^D(P)$. Assume that there is a minimal model N of $\pi^D(P)$ such that $N \subset M$. Since N is not a model of P, N does not satisfy the clause (7.1). Then $N \models \Gamma$, $N \models A$, and $N \not\models \Sigma$. As a minimal model N of $\pi^D(P)$ implies A, it follows from Lemma 7.3 that there is a clause C of the form (7.2) or (7.3) in $\pi^D(P)$ such that C contains A in its head. (i) Suppose first that C is of the form (7.2). Then $N \models A$ implies $N \setminus \{A\} \models \Gamma_i$ and $N \setminus \{A\} \not\models \Sigma_i$ (by Lemma 7.3). Here $N \setminus \{A\} \models \Gamma_i$ implies $N \models \Gamma_i$. Besides, the disjunctive head $A \vee \Sigma_i$ is assumed to be already factored, then Σ_i does not include A. Thus $N \setminus \{A\} \not\models \Sigma_i$ also implies $N \not\models \Sigma_i$. In this case, however, N does not satisfy the clause (7.3). This contradicts the assumption that N is a model of $\pi^D(P)$. (ii) Next suppose that C is of the form (7.3) such that $\Sigma = A \vee \Sigma'$. Then $N \models A$ implies $N \models \Sigma$, which contradicts the fact $N \not\models \Sigma$. Hence, M is also a minimal model of $\pi^D(P)$.

($\supseteq$) Let M be a minimal model of $\pi^D(P)$. If M is not a model of P, M does not satisfy the clause (7.1). In this case, $M \not\models \Sigma$, $M \models A$, and $M \models \Gamma$. Since a minimal model M of $\pi^D(P)$ implies A, it follows from Lemma 7.3 that there is a clause C of the form (7.2) or (7.3) in $\pi^D(P)$ such that C contains A in its head. When C is of the form (7.2), $M \models A$ implies $M \models \Gamma_i$ and $M \not\models \Sigma_i$ (by Lemma 7.3 and the discussion presented above). Thus M does not satisfy the corresponding clause (7.3), which contradicts the assumption that M is a model of $\pi^D(P)$. Else when C is of the form (7.3) such that $\Sigma = A \vee \Sigma'$, $M \models A$ implies $M \models \Sigma$, which contradicts the fact $M \not\models \Sigma$. Hence M is a model of P. Next assume that

there is a minimal model N of P such that $N \subset M$. By ($\subseteq$), N is also a minimal model of $\pi^D(P)$, but this is impossible since M is a minimal model of $\pi^D(P)$. □

Corollary 7.5 Let P be a positive disjunctive program. Then P is inconsistent iff $\pi^D(P)$ is inconsistent. □

Example 7.3 (cont. from Example 7.2) Given the program P, its disjunctive partial deduction $\pi^D_{\{c \leftarrow a;a\}}(P)$ becomes

$$\pi^D_{\{c \leftarrow a;a\}}(P) = \{\ a \vee b \leftarrow,\quad a \leftarrow d,\quad c \leftarrow d,\quad b \vee c \leftarrow\ \},$$

and $\mathcal{MM}_{\pi^D_{\{c \leftarrow a;a\}}(P)} = \{\{a, c\}, \{b\}\}$, which is exactly the same as $\mathcal{MM}_P$. □

7.3 Partial Deduction of Normal Disjunctive Programs

In this section, we extend disjunctive partial deduction to normal disjunctive programs.

7.3.1 Disjunctive Partial Deduction of Normal Disjunctive Programs

The definition of disjunctive partial deduction for normal disjunctive programs is the same as Definition 7.1, except that in this case each clause possibly contains default negation.

Example 7.4 Let P be the normal disjunctive program:

$$P = \{\ a \vee b \leftarrow not\ c,\quad a \leftarrow d,\quad c \leftarrow a\ \}.$$

Then disjunctive partial deduction of P with respect to $c \leftarrow a$ on a becomes

$$\pi^D_{\{c \leftarrow a;a\}}(P) = \{\ a \vee b \leftarrow not\ c,\quad a \leftarrow d,\quad c \leftarrow d,\quad b \vee c \leftarrow not\ c\ \}.\quad \square$$

As shown in the above example, disjunctive partial deduction is not affected by the presence of default negation in a program. Thus we can directly apply previously defined disjunctive partial deduction to normal disjunctive programs and the following result holds.

Theorem 7.6 Let P be a normal disjunctive program and $\mathcal{ST}_P$ be the set of all stable models of P. Then $\mathcal{ST}_P = \mathcal{ST}_{\pi^D(P)}$.

Proof: Let M be a stable model of P. Then M is a minimal model of P^M. Since P^M is a positive disjunctive program, by Theorem 7.4, M is also a minimal model of $\pi^D(P^M)$. Now let us consider the clauses:

$$\Sigma \leftarrow A \wedge \Gamma \wedge not\, \Gamma' \quad (*)$$

and

$$A \vee \Sigma_i \leftarrow \Gamma_i \wedge not\, \Gamma_i' \quad (1 \leq i \leq k) \quad (\dagger)$$

in P, where $not\, \Gamma'$ is the conjunction of default-negation formulas in the body.

(i) If $M \not\models \Gamma'$ and $M \not\models \Gamma_i'$ for some i $(1 \leq i \leq k)$, the clauses:

$$\Sigma \leftarrow A \wedge \Gamma \quad (*')$$

and

$$A \vee \Sigma_i \leftarrow \Gamma_i \quad (\dagger')$$

are in P^M. From these clauses, disjunctive partial deduction generates the clauses:

$$\Sigma \vee \Sigma_i \leftarrow \Gamma \wedge \Gamma_i \quad (\ddagger')$$

in $\pi^D(P^M)$. On the other hand, from $(*)$ and $(\dagger)$ in P, there are the clauses:

$$\Sigma \vee \Sigma_i \leftarrow \Gamma \wedge \Gamma_i \wedge not\, \Gamma' \wedge not\, \Gamma_i' \quad (\ddagger)$$

in $\pi^D(P)$, which become $(\ddagger')$ in $\pi^D(P)^M$.

(ii) Else if $M \models \Gamma'$ or $M \models \Gamma_i'$ for any i $(1 \leq i \leq k)$, the clauses $(*)$ or $(\dagger)$ is respectively eliminated in P^M. Then the clauses $(\ddagger')$ are not included in $\pi^D(P^M)$. In this case, each clause $(\ddagger)$ in $\pi^D(P)$ is also eliminated in $\pi^D(P)^M$.

Thus, there is a one-to-one correspondence between the clauses in $\pi^D(P^M)$ and the clauses in $\pi^D(P)^M$, hence $\pi^D(P^M) = \pi^D(P)^M$. Therefore M is also a minimal model of $\pi^D(P)^M$, and a stable model of $\pi^D(P)$.

The converse is also shown in the same manner. □

Corollary 7.7 Let P be a normal disjunctive program. Then P is incoherent iff $\pi^D(P)$ is incoherent. □

The above theorem also implies that in normal logic programs, normal partial deduction preserves Gelfond and Lifschitz's stable model semantics.

Corollary 7.8 Let P be a normal logic program. Then $\mathcal{ST}_P = \mathcal{ST}_{\pi^N(P)}$. □

The above result is also presented in [Seki, 1990].

7.3.2 Connections between Normal and Disjunctive Partial Deduction

In this section, we consider connections between normal and disjunctive partial deduction. We first give a sufficient condition such that normal partial deduction preserves the meaning of disjunctive programs.

Theorem 7.9 Let P be a normal disjunctive program and C be a clause of the form $\Sigma \leftarrow A \wedge \Gamma$ from P. If A does not appear in the head of any disjunctive clause in P, then $\mathcal{ST}_P = \mathcal{ST}_{\pi^N_{(C,A)}(P)}$. That is, normal partial deduction of P with respect to C on A preserves the disjunctive stable model semantics.

Proof: In this case, disjunctive partial deduction coincides with normal one, hence the result follows from Theorem 7.6. □

Next we present a method to compute disjunctive partial deduction in terms of normal partial deduction.

Definition 7.2 Let P be a normal disjunctive program. Then the *nlp-transformation* transforms P into the normal logic program $\eta(P)$ which is obtained from P by replacing each disjunctive clause:

$$C:\ A_1 \vee \ldots \vee A_l \leftarrow \Gamma \tag{7.4}$$

with l normal clauses:

$$C_i^-:\ A_i \leftarrow \Gamma \wedge A_1^- \wedge \ldots \wedge A_{i-1}^- \wedge A_{i+1}^- \wedge \ldots \wedge A_l^- \quad (1 \leq i \leq l). \tag{7.5}$$

where each A_j^- is a new atom introduced for each A_j.
In particular, $C = C_i^-$ if $l \leq 1$. □

Now we show that disjunctive partial deduction of a normal disjunctive program P with respect to a clause C is obtained through normal partial deduction of $\eta(P)$ with respect to each C_i^-. In the following, the function η^{-1} is the reverse transformation which shifts each atom A_j^- appearing in the body of each clause in a program to the atom A_i in the head of the clause. Also Σ^- means $A_1^- \wedge \ldots \wedge A_l^-$ where $\Sigma = A_1 \vee \ldots \vee A_l$.

Theorem 7.10 Let P be a normal disjunctive program. Then,

$$\pi^D_{\{C;A\}}(P) = \eta^{-1}(\pi^N_{\{C_i^-;A\}}(\eta(P)))$$

where $\pi^N_{\{C_i^-;A\}}(\eta(P))$ means normal partial deduction of $\eta(P)$ with respect to each normal clause C_i^- on A.

Proof: Corresponding to the clauses (7.1) and (7.2) in P, there are the clauses:

$$A' \leftarrow A \wedge \Gamma \wedge \Sigma'^- \quad (\text{where } \Sigma = \Sigma' \vee A') \quad (*)$$

and

$$A \leftarrow \Gamma_i \wedge \Sigma_i^- \quad (1 \leq i \leq k) \quad (\dagger)$$

in $\eta(P)$, respectively. Then the clauses:

$$A' \leftarrow \Gamma \wedge \Gamma_i \wedge \Sigma'^- \wedge \Sigma_i^- \quad (\ddagger)$$

are obtained from $(*)$ and $(\dagger)$ by normal partial deduction in $\eta(P)$. In this case, by the reverse transformation η^{-1}, each clause of the form $(\ddagger)$ becomes a disjunctive clause of the form (7.3). Hence, $\pi^D_{\{C;A\}}(P) = \eta^{-1}(\pi^N_{\{C_i^-;A\}}(\eta(P)))$. □

Example 7.5 Let P be the program:

$$P = \{\ a \vee b \leftarrow,\quad a \leftarrow b,\quad b \leftarrow a\ \}.$$

Then,

$$\pi^D_{\{a \leftarrow b;b\}}(P) = \{\ a \vee b \leftarrow,\quad a \leftarrow,\quad a \leftarrow a,\quad b \leftarrow a\ \}.$$

On the other hand, the *nlp*-transformation of P becomes

$$\eta(P) = \{\ a \leftarrow b^-,\quad b \leftarrow a^-,\quad a \leftarrow b,\quad b \leftarrow a\ \},$$

and

$$\pi^N_{\{a \leftarrow b;b\}}(\eta(P)) = \{\ a \leftarrow b^-,\quad b \leftarrow a^-,\quad a \leftarrow a^-,\quad a \leftarrow a,\quad b \leftarrow a\ \}.$$

Thus,

$$\eta^{-1}(\pi^N_{\{a \leftarrow b;b\}}(\eta(P))) = \{\ a \vee b \leftarrow,\quad a \leftarrow,\quad a \leftarrow a,\quad b \leftarrow a\ \}.$$

Therefore, $\pi^D_{\{a \leftarrow b;b\}}(P) = \eta^{-1}(\pi^N_{\{a \leftarrow b;b\}}(\eta(P)))$. □

The above theorem presents that disjunctive partial deduction $\pi^D_{\{C;A\}}(P)$ is obtained by the transformation sequence: $P \rightarrow \eta(P) \rightarrow \pi^N_{\{C_i^-;A\}}(\eta(P)) \rightarrow \eta^{-1}(\pi^N_{\{C_i^-;A\}}(\eta(P)))$. That is, together with the *nlp*-transformation, normal partial deduction can also be used for normal disjunctive programs.

7.3.3 Preservation of the Possible Model Semantics

We consider partial deduction for disjunctive programs under the possible model semantics. Unfortunately, disjunctive partial deduction does not preserve the possible model semantics in general.

Example 7.6 Let P be the disjunctive program:

$$P = \{\ a \vee b \leftarrow,\quad c \leftarrow b\ \}.$$

Then disjunctive partial deduction of P with respect to $c \leftarrow b$ on b becomes

$$\pi^D_{\{c \leftarrow b;b\}}(P) = \{\ a \vee b \leftarrow,\quad a \vee c \leftarrow\ \}.$$

In this case, $\mathcal{PM}_P = \{\{a\}, \{b,c\}, \{a,b,c\}\}$, while the residual program has the possible models $\{a,b\}$ and $\{a,c\}$ in addition to $\mathcal{PM}_P$. □

In the above example, a dependency relation between c and b is not expressed after partial deduction. This observation tells us that resolution-based disjunctive partial deduction often fails to preserve syntax-dependent logic programming semantics. In fact, the above example shows that even the supported models of P are not preserved during partial deduction; $\{a,c\}$ is a supported model of the residual program but it is not a supported model of P.

Then we first consider a sufficient condition for preserving the possible model semantics. A result similar to Theorem 7.9 holds.

Theorem 7.11 Let P be a normal disjnnctive program and C be a clause of the form $\Sigma \leftarrow A \wedge \Gamma$ from P. If A does not appear in the head of any disjunctive clause in P, then $\mathcal{PM}_P = \mathcal{PM}_{\pi^D_{\{C,A\}}(P)}$.

Proof: In this case, stable models of each split program of P are preserved, hence the result follows. □

Next we present a method of partial deduction for preserving possible models in general. We have already seen in Section 3.6 that possible models of any disjunctive program are expressed by stable models of a normal logic program obtained by the *pm*-transformation. Then, we compute partial deduction for the possible model semantics in terms of normal partial deduction in such a transformed program.

The following result follows from Theorem 3.30 and Corollary 7.8.

Theorem 7.12 Let P be a normal disjunctive program and $\wp(P)$ be a normal logic program obtained by *pm*-transformation. Then $\mathcal{PM}_P = \mathcal{ST}_{\pi^N(\wp(P))} \cap \mathcal{HB}_P$. □

7.4 Goal-Oriented Partial Deduction

In this section, we present goal-oriented partial deduction in disjunctive programs. Goal-oriented partial deduction specializes a program with respect to a given goal, which is useful to optimize programs for query-answering. Lloyd and Shepherdson [1991] discuss a framework of goal-oriented partial evaluation for normal logic programs with respect to SLDNF proof procedures. In our framework, goal-oriented partial deduction is presented as follows.

Let us consider a query of the form:

$$Q:\ Q(\mathbf{x}) \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\ B_{m+1} \wedge \ldots \wedge not\ B_n \tag{7.6}$$

where $Q(\mathbf{x})$ is a new atom not appearing elsewhere in a program and $\mathbf{x}$ represents variables appearing in the body of the clause.

Then, given a normal disjunctive program P, partial deduction of P with respect to Q is defined as $\pi^D_{\{Q;B_i\}}(P_Q)$ where B_i is any atom occurring positively in the body of Q and P_Q is the program $P \cup \{Q\}$. When a query contains variables, we consider partial deduction with respect to its ground instances.

As introduced in Section 3.5, an answer to a query is defined as a ground substitution σ for variables in $Q(\mathbf{x})$.

A query Q is *true* in P under the disjunctive stable model semantics if for every stable model I of P_Q there is an answer σ such that $Q(\mathbf{x})\sigma$ is included in I. Else if for some stable model I of P_Q there is an answer σ such that $Q(\mathbf{x})\sigma$ is included in I, the query is *possibly true*. Otherwise, if there is no such answer, the query is *false*. By Theorem 7.6, the following results hold.

Theorem 7.13 Let P be a normal disjunctive program and Q be a query. Then, under the disjunctive stable model semantics,

(i) Q is true in P iff Q is true in $\pi^D_{\{Q;B_i\}}(P_Q)$.

(ii) Q is possibly true in P iff Q is possibly true in $\pi^D_{\{Q;B_i\}}(P_Q)$.

(iii) Q is false in P iff Q is false in $\pi^D_{\{Q;B_i\}}(P_Q)$. □

Example 7.7 Let P be the program:

$$\{\ p(a) \vee p(b) \leftarrow\ \},$$

in which the query $Q:\ q(x) \leftarrow p(x)$ is true. Then,

$$\pi^D_{\{Q;p(x)\}}(P_Q) = \{\ q(a) \vee p(b) \leftarrow,\quad q(b) \vee p(a) \leftarrow,\quad q(a) \vee q(b) \leftarrow,\quad p(a) \vee p(b) \leftarrow\ \}$$

and Q is also true in $\pi^D_{\{Q;p(x)\}}(P_Q)$ under the disjunctive stable model semantics. □

Note that in the above example, we assume that the ground queries $q(a) \leftarrow p(a)$ and $q(b) \leftarrow p(b)$ are unfolded consecutively in the program. That is, $\pi^D_{\{Q;p(x)\}}(P_Q)$ means $\pi^D_{\{Q;p(b)\}}(\pi^D_{\{Q;p(a)\}}(P \cup \{\ q(a) \leftarrow p(a),\ \ q(b) \leftarrow p(b)\ \}))$. In this case, the order of unfolding does not affect the result of partial deduction since each partial deduction preserves the stable models of the program P_Q.

Using the technique presented in the previous section, corresponding results also hold for goal-oriented partial deduction under the possible model semantics.

7.5 Discussion

So far, we have considered partial deduction of propositional programs, while disjunctive partial deduction is also directly extended to programs containing variables.

Let P be a normal disjunctive program and C be a clause in P of the form:

$$C:\ \Sigma \leftarrow A \wedge \Gamma.$$

Then, for any clause

$$C_i:\ A_i \vee \Sigma_i \leftarrow \Gamma_i\ \ (1 \le i \le k)$$

from P such that $A\sigma_i = A_i\sigma_i$ holds with an mgu σ_i, the following clauses

$$C'_i:\ (\Sigma \vee \Sigma_i \leftarrow \Gamma \wedge \Gamma_i)\sigma_i$$

are in the residual program $\pi^D_{\{C;A\}}(P) = (P \setminus \{C\}) \cup \{C'_1, \ldots, C'_k\}$.

Correctness of such disjunctive partial deduction with respect to the disjunctive stable model semantics is proved in a similar way to the propositional case.

Disjunctive partial deduction is implemented on the bottom-up model generation procedure presented in Section 3.5, and it preserves the disjunctive stable model semantics in range-restricted function-free normal disjunctive programs. On the other hand, a transformation from disjunctive partial deduction to normal partial deduction presented in the previous section enables us to use a partial evaluator for normal logic programs also as a procedure for disjunctive partial deduction. Partial deduction under the possible model semantics is also executed using a partial evaluator of normal logic programs with respect to the stable model semantics.

In the context of deductive databases, a program usually consists of a large amount of data (called an *extensional database*) and a comparatively small set of derivation rules (called an *intensional database*). For query processing in such a database, it is effective to partially evaluate a query in the intensional database at first, and subsequently evaluate it in the extensional database [Gallaire et al., 1984]. Such a

technique can also be realized in our goal-oriented partial deduction presented in Section 7.4. Related topics are discussed in [Sakama and Itoh, 1988] for optimizing queries in deductive databases.

The partial deduction technique presented in this chapter is also directly applicable to extended disjunctive programs. Moreover, since positive disjunctive programs are identified with first-order theories, disjunctive partial deduction has potential application to first-order theorem provers. Recently, Brass and Dix [1994] independently developed a partial deduction technique for disjunctive programs which is equivalent to ours. They discuss several abstract properties of disjunctive programs and conclude partial deduction as one of the fundamental properties that logic programming semantics should satisfy.

7.6 Summary

In this chapter, we have presented a method of partial deduction for disjunctive programs. We first showed that normal partial deduction is not applicable to disjunctive programs in its present form. Then we introduced disjunctive partial deduction for disjunctive programs, which is a natural extension of normal partial deduction for normal logic programs. Disjunctive partial deduction was shown to preserve the minimal model semantics of positive disjunctive programs, and the disjunctive stable model semantics of normal disjunctive programs. We also showed a method of translating disjunctive partial deduction into normal partial deduction, and presented an application to goal-oriented partial deduction for query optimization.

Chapter 8

Conclusion

8.1 Summary and Contributions

In this dissertation, we have studied theoretical aspects of disjunctive logic programming from various viewpoints.

It has been considered that the principle of minimality is a basic criterion for any rational semantics of logic programming and commonsense reasoning in AI. However, it was also noticed that this principle is not always appropriate in the presence of indefinite information in a program. Then we started to develop a new theoretical framework of disjunctive logic programs, which *violates* this common principle. To this end, in Chapter 3 we proposed a new declarative semantics of disjunctive logic programs called the possible model semantics. Due to its non-minimal property, the possible model semantics can freely specify both inclusive and exclusive disjunctions in a program, and provides a flexible negative inference mechanism under the closed world assumption. The possible model semantics was also characterized by a new fixpoint semantics of disjunctive logic programs, and a bottom-up proof procedure for computing possible models was presented. The advantage of the possible model semantics lies not only in its representational power, but also in its computational complexity. That is, computation of possible models does not need an extra mechanism for minimality-checking, which makes computing the possible model semantics much easier than the minimal-model based semantics. We have verified this fact by comparing the computational complexity of each semantics.

Logic programming semantics is known to be closely related to nonmonotonic formalisms in AI. There have been studied methods of representing normal logic programs in terms of nonmonotonic formalisms, and their extensions to disjunctive logic programs were also proposed by several researchers. In Chapter 4, however, we have

pointed out that the previously studied result on default translations of disjunctive logic programs is incorrect. Then we introduced an alternative correct transformation from disjunctive logic programs to default theories, and showed that the disjunctive stable model semantics is characterized in terms of default extensions. The result indicates that Reiter's default logic still works well to characterize disjunctive logic programs, which breaks the folklore that Reiter's default logic is inappropriate to characterize disjunctive logic programs in general. In fact, Gelfond et al.'s disjunctive default logic is proposed to treat disjunctions "properly" in a default theory, while we have shown that Reiter's default logic has the same expressiveness as disjunctive default logic to characterize the semantics of disjunctive logic programs. Disjunctive logic programs were also characterized by autoepistemic logic and circumscription. Moreover, we have shown that the possible model semantics of disjunctive logic programs is expressed by the non-minimal feature of autoepistemic expansions.

Abduction is also a form of commonsense reasoning in AI, and its application to logic programming is known as abductive logic programming. Disjunctive logic programs and abductive logic programs are two extensions of logic programming which provide frameworks for reasoning with incomplete information, while little attention has been paid for their interrelations. Then, in Chapter 5, we revealed a close relationship between disjunctive logic programs and abductive logic programs. It was shown that the possible model semantics of disjunctive logic programs is essentially equivalent to the generalized stable models of abductive logic programs. This fact indicates that disjunctive logic programs and abductive logic programs are just different ways of looking at the same problem if we choose the appropriate semantics. Moreover, the possible model semantics is useful not only for disjunctive logic programs but also for abductive logic programs, and establishes links between each framework. The usefulness of the possible model semantics in abductive logic programming was also verified from the computational complexity viewpoint.

Another important issue for commonsense reasoning in logic programming is the treatment of inconsistent information in a program. In the context of extended logic programs, a program may become inconsistent in the presence of explicit negation. However, classical logic programming framework is useless in the presence of inconsistent information, and paraconsistent extensions are needed in order to treat inconsistent information properly in a program. In Chapter 6, we then proposed paraconsistent frameworks for disjunctive logic programs which can cope with inconsistent information as well as indefinite information in a program. The paraconsistent stable and possible model semantics were introduced for extended disjunctive programs, and those semantics were shown to be useful compared with the answer set semantics.

Each paraconsistent semantics was also characterized by the fixpoint semantics of extended disjunctive programs, and various methods for reasoning with inconsistent information were discussed.

Disjunctive logic programs increase expressive power of logic programming, while their computation are generally more expensive than normal logic programs. Then, in Chapter 7, we developed partial deduction techniques for optimizing disjunctive logic programs. We first argued that normal partial deduction for logic programs is not directly applicable to disjunctive logic programs, then introduced a new partial deduction method for disjunctive logic programs. The proposed disjunctive partial deduction was shown to preserve the minimal model semantics of positive disjunctive programs and the disjunctive stable models semantics of normal disjunctive programs. We also showed a method of translating disjunctive partial deduction into normal partial deduction, and presented a method of using normal partial deduction for the possible model semantics. An application to goal-oriented partial deduction is also presented for query optimization in disjunctive deductive databases.

In this dissertation, we have discussed semantic issues for various extensions of logic programming. Interrelations between those extensions are presented in Figure 8.1.

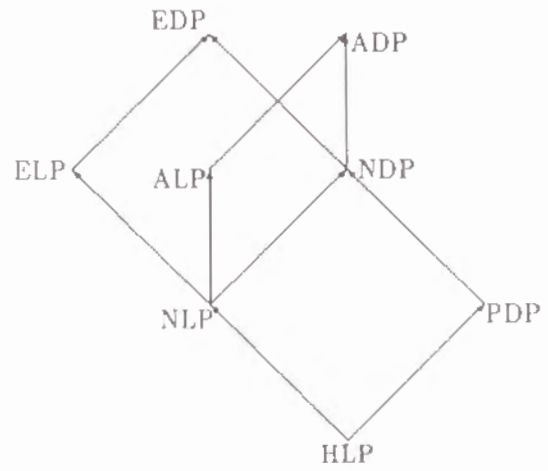
Now we know the following relationships between each class of programs.

- Normal disjunctive programs are reducible to positive disjunctive program by the epistemic transformation presented in Section 3.4. This transformation preserves both the disjunctive stable model semantics and the possible model semantics.

- Normal disjunctive programs are reducible to normal logic programs preserving the possible model semantics by the *pm*-transformation presented in Section 3.6. By contrast, under the disjunctive stable model semantics, such a transformation is most unlikely possible in polynomial time.

- Abductive logic programs are transferable to normal disjunctive programs and vice versa under the possible model semantics. Also abductive disjunctive programs are reducible to normal disjunctive programs under the disjunctive stable model semantics and the possible model semantics (Chapter 5).

- Extended disjunctive programs and extended logic programs are respectively reducible to normal disjunctive programs and normal logic programs by considering their positive forms. Each semantics for extended programs are expressed by the corresponding semantics of normal programs (Chapter 6).



HLP: Horn logic programs, NLP: normal logic programs, PDP: positive disjunctive programs, NDP: normal disjunctive programs, ELP: extended logic programs, EDP: extended disjunctive programs, ALP: abductive logic programs, ADP: abductive disjunctive programs.

Figure 8.1: Extensions of Logic Programming

The first fact presents that default negation can be represented by disjunctions, while the second fact presents that the converse is also true under the possible model semantics. The third fact presents that abductive hypotheses are identified with disjunctions under the possible model semantics, and abductive disjunctive programs do not increase expressive power of normal disjunctive programs. The fourth fact presents that classical negation in extended programs can be interpreted within the frameworks of normal programs.

These observations indicate a somewhat surprising result that *all "extensions" of logic programming are essentially equivalent under the possible model semantics. That is, default negation, disjunction, and abducibles can be used interchangeably under the possible model semantics, and classical negation is nothing but "syntax-sugar".*

This fact suggests that the possible model semantics provides a unifying framework for the semantics of logic programs, and its potential expressiveness would contribute to enrich our perspectives of logic programming as a theoretical tool for commonsense reasoning in artificial intelligence.

8.2 Future Research

In this dissertation, we have discussed various extensions of logic programming and their theoretical frameworks. Syntactically speaking, further extensions besides those presented in the dissertation are considered. One extension is incorporating classical negation in abductive logic/disjunctive programs. *Abductive extended logic programs* and *abductive extended disjunctive programs* are extensions of abductive logic programs and abductive disjunctive programs respectively, which are obtained by replacing normal logic/disjunctive programs with extended logic/disjunctive programs in each abductive framework. However, such frameworks are reducible to normal abductive logic/disjunctive programs by considering their positive forms, so it is easy to extend the results presented in Chapter 5 to abductive extended logic/disjunctive programs. Inoue and Sakama [1993] have introduced a fixpoint semantics of abductive extended disjunctive programs and shown that abductive extended disjunctive programs are transferable to extended disjunctive programs under the answer set semantics.

Another extension is to incorporate default negation in the head as well as in the body of each clause in extended disjunctive programs. Such an extension of logic programming is known as logic programs with *"positive not"*. A unique feature of such programs is that answer sets of those programs are not necessarily minimal, however, its application to knowledge representation was remained open [Lifschitz and Woo, 1992]. Inoue and Sakama [1994] have recently shown that such non-minimal answer sets are useful to characterize abductive reasoning and inclusive disjunctions in logic programming. Moreover, they show that the possible model semantics plays an important role for such characterizations. This extension of logic programming is fairly new, and further applications of such programs are remained to be seen. However, since the framework is closely related to the possible model semantics, further investigation would also exploit new applications of the possible model semantics.

Considering the relationship to nonmonotonic reasoning, we have shown in Chapter 4 that disjunctive logic programs are translated into nonmonotonic formalisms in AI. Since abductive logic/disjunctive programs are transferable to disjunctive logic programs as presented in Chapter 5, by combining these two transformations, we can also relate abductive logic programming to each nonmonotonic formalism. From the computational complexity viewpoint, it is known that disjunctive stable model semantics and those nonmonotonic formalisms are interrelated at the second level of the polynomial hierarchy [Eiter and Gottlob, 1993b]. This fact implies that reasoning tasks in each nonmonotonic formalism can also be efficiently realized in disjunctive

logic programs. In other words, disjunctive logic programs have potential applications for nonmonotonic reasoning in AI.

On the other hand, we have shown that non-minimal nature of autoepistemic logic is useful to characterize inclusive disjunctions in knowledge representation. It is shown in [Inoue and Sakama, 1994] that such non-minimal autoepistemic expansions are also useful for characterizing abduction. These facts suggest that the principle of minimality is not a dominant rule for commonsense reasoning in AI any more. So it is interesting to seek further applications of non-minimal nonmonotonic reasoning, which would open new perspectives of nonmonotonic formalisms as knowledge representation tools.

As for the procedural aspects of disjunctive logic programs, in Section 3.5 we have presented a bottom-up proof procedure which is sound and complete with respect to the possible model semantics/disjunctive stable model semantics for function-free range-restricted normal disjunctive programs. However, the proposed bottom-up proof procedure is rather naive for query-answering, and in the presence of huge databases, it might need some optimization techniques as presented in [Bancilhon and Ramakrishnan, 1988]. For instance, partial deduction presented in Chapter 7 is one of such optimization techniques, and it would be used to reduce search space for a given query. In this dissertation, we have not presented a top-down proof procedure for disjunctive logic programs since it is generally inefficient. However, as discussed in Section 3.7, concerning the possible model semantics a top-down proof procedure is realized using a proof procedure for the stable model semantics in *pm*-transformed normal logic programs.

From the standpoint of deductive databases, *integrity checking* and *view update* in disjunctive logic programs are also important. We have considered the meaning of a disjunctive logic program as the collection of all possible models satisfying integrity constraints. Such selection is done during the bottom-up computation of possible models and any model violating integrity constraints is pruned away. Thus, if the procedure generates no possible model, we know that a database violates integrity constraints. Note that we consider integrity constraints in the form of negative clauses, while more general form of integrity constraints are transformed into negative clauses as presented in [Sadri and Kowalski, 1988].

There are difficulties in view update in disjunctive deductive databases. Without disjunctive information, view update is usually achieved by translating an update request on a virtual relation into a real update on the underlying database relations. For example, in a database containing the clause $p(x) \leftarrow q(x)$ where p is a virtual relation and q is a real database relation, the addition of the virtual fact $p(a)$ is translated into the addition of the real fact $q(a)$. Then $p(a)$ is implied in the updated database. However, given the disjunctive clause $p(x) \vee r(x) \leftarrow q(x)$, it is not so easy to translate the addition of $p(a)$ into the real update on $q(x)$. This is because the addition of $q(a)$ in this case does not necessarily imply the truth of $p(a)$. Several problems arise in updating indefinite knowledge bases and further investigation is needed.

Last but not least, practical applications of disjunctive logic programming are also very important. In this dissertation, we are mainly concerned with the theoretical aspects of disjunctive logic programming, however, we consider that disjunctive logic programming has promising applications due to its rich expressiveness. For instance, in Section 5 we have presented an equivalence relationship between disjunctive and abductive logic programs. This fact implies that applications of abductive logic programming, such as diagnosis and planning, can also become applications of disjunctive logic programming. On the other hand, recent studies in artificial intelligence recognize the need of standardized knowledge representation for developing reusable and sharable knowledge bases [Neches et al., 1991]. Such technologies are important since they would greatly reduce the cost of designing and maintaining knowledge bases. Of course it might be difficult to develop a single common language which is useful for different kinds of multi-purpose knowledge bases. However, logic programming has potential possibilities to serve as an archetype of the common language because it provides a universal framework based on mathematical logic. Moreover, since logic programming has nice relations with databases and nonmonotonic reasoning, it enables us to develop knowledge bases combining existing database and AI technologies.

Thus we believe that logic programming and disjunctive logic programming play important roles in knowledge representation and artificial intelligence in the next generation, and we hope that the studies presented in this dissertation will contribute to further development of research in the fields.

List of Publications

[Sakama and Itoh, 1988] Sakama, C. and Itoh, H., Partial Evaluation of Queries in Deductive Databases, *New Generation Computing* 6(2&3), Ohmsha and Springer-Verlag, 249-258, 1988.

[Sakama and Itoh, 1988] Sakama, C. and Itoh, H., Handling Knowledge by its Representative, *Proceedings of the 2nd International Conference on Expert Database Systems*, Benjamin/Cummings, 551-565, 1988.

[Sakama and Okumura, 1988] Sakama, C. and Okumura, A., Nonmonotonic Parallel Inheritance Network, *Logic Programming '88, Proceedings of the 7th Conference*, Lecture Notes in Artificial Intelligence 383, Springer-Verlag, 53-66, 1988.

[Sakama, 1989] Sakama, C., Possible Model Semantics for Disjunctive Databases, *Proceedings of the 1st International Conference on Deductive and Object-Oriented Databases*, North-Holland, 369-383, 1989.

[Sakama, 1992] Sakama, C., Extended Well-founded Semantics for Paraconsistent Logic Programs, *Proceedings of the International Conference on Fifth Generation Computer Systems*, ICOT, 592-599, 1992.

[Inoue and Sakama, 1992] Inoue, K. and Sakama, C., A Uniform Approach to Fixpoint Characterization of Disjunctive and General Logic Programs, ICOT Technical Report TR-817, 1992.

[Inoue and Sakama, 1993] Inoue, K. and Sakama, C., Transforming Abductive Logic Programs to Disjunctive Programs, *Proceedings of the 10th International Conference on Logic Programming*, MIT Press, 335-353, 1993.

[Sakama and Inoue, 1993a] Sakama, C. and Inoue, K., Negation in Disjunctive Logic Programs, *Proceedings of the 10th International Conference on Logic Programming*, MIT Press, 703-719, 1993.

[Sakama and Inoue, 1993b] Sakama, C. and Inoue, K., Relating Disjunctive Logic Programs to Default Theories, *Proceedings of the 2nd International Workshop on Logic Programming and Nonmonotonic Reasoning*, MIT Press, 266-282, 1993.

[Inoue and Sakama, 1994] Inoue, K. and Sakama, C., On Positive Occurrences of Negation as Failure, *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann, 293-304, 1994.

[Sakama and Inoue, 1994a] Sakama, C. and Inoue, K., On the Equivalence between Disjunctive and Abductive Logic Programs, *Proceedings of the 11th International Conference on Logic Programming*, MIT Press, 489-503, 1994.

[Sakama and Inoue, 1994b] Sakama, C. and Inoue, K., Paraconsistent Stable Semantics for Extended Disjunctive Programs, *Journal of Logic and Computation*, Oxford University Press, to appear, 1994.

[Sakama and Inoue, 1994c] Sakama, C. and Inoue, K., An Alternative Approach to the Semantics of Disjunctive Logic Programs and Deductive Databases, *Journal of Automated Reasoning*, Kluwer Academic, to appear, 1994.

[Sakama and Seki, 1994] Sakama, C. and Seki, H., Partial Deduction of Disjunctive Logic Programs: A Declarative Approach, *Proceedings of the 4th International Workshop on Logic Program Synthesis and Transformation*, Lecture Notes in Computer Science, Springer-Verlag, to appear, 1994.

Bibliography

[Alferes and Pereira, 1992] Alferes, J. J. and Pereira, L. M., On Logic Program Semantics with Two Kinds of Negation, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, MIT Press, 574-588, 1992.

[Alferes et al., 1994] Alferes, J. J., Damasio, C. V. and Pereira, L. M., Top-Down Query Evaluation for Well-Founded Semantics with Explicit Negation, *Proceedings of the 11th European Conference on Artificial Intelligence*, John Wiley&Sons, 1994.

[Apt and van Emden, 1982] Apt, K. R. and van Emden, M. H., Contributions to the Theory of Logic Programming, *Journal of the ACM* 29(3), 841-862, 1982.

[Apt et al., 1988] Apt, K. R., Blair, H. A. and Walker, A., Towards a Theory of Declarative Knowledge, in [Minker, 1988], 89-148, 1988.

[Apt, 1990] Apt, K. R., Logic Programming, in *Handbook of Theoretical Computer Science*, vol. B, J. van Leeuwen (ed.), Elsevier Science Publishers B. V., North-Holland, 495-574, 1990.

[Arruda, 1980] Arruda, A. I., A Survey of Paraconsistent Logic, *Mathematics Logic in Latin America*, A. I. Arruda, R. Chuaqui, and N. C. A. da Costa (eds.), North-Holland, 1-41, 1980.

[Bancilhon and Ramakrishnan, 1988] Bancilhon, F. and Ramakrishnan, R., Performance Evaluation of Data Intensive Logic Programs, in [Minker, 1988], 439-517, 1988.

[Baral et al., 1992a] Baral, C., Lobo, J. and Minker, J., Generalized Disjunctive Well-Founded Semantics for Logic Programs, *Aanals of Mathematics and Artificial Intelligence* 5, 89-132, 1992.

[Baral et al., 1992b] Baral, C., Kraus, S., Minker, J., and Subrahmanian, V. S., Combining Knowledge Bases Consisting of First-Order Theories, *Computational Intelligence* 8(1), 45-71, 1992.

[Baral and Subrahmanian, 1992] Baral, C. and Subrahmanian, V. S., Stable and Extension Class Theory for Logic Programs and Default Logics, *Journal of Automated Reasoning* 8(3), 345-366, 1992.

[Baral and Subrahmanian, 1993] Baral, C. and Subrahmanian, V. S., Dualities between Alternative Semantics for Logic Programming and Nonmonotonic Reasoning, *Journal of Automated Reasoning* 10(3), 399-420, 1993.

[Baral and Gelfond, 1994] Baral, C. and Gelfond, M., Logic Programming and Knowledge Representation, *Journal of Logic Programming* 19/20, 73-148, 1994.

[Bell et al., 1993] Bell, C., Nerode, A., Ng, R., and Subrahmanian, V. S., Implementing Stable Semantics by Linear Programming, in [Pereira and Nerode, 1993], 23-42, 1993.

[Belnap, 1975] Belnap, N. D., A Useful Four-Valued Logic, in *Modern Uses of Multiple-Valued Logic*, J. M. Dunn and G. Epstein (eds.), Reidel Publishing, 8-37, 1975.

[Ben-Eliyahu and Dechter, 1992] Ben-Eliyahu, R. and Dechter, R., Propositional Semantics for Disjunctive Logic Programs, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, MIT Press, 813-827, 1992.

[Bidoit and Hull, 1986] Bidoit, N. and Hull, R., Positivism vs. Minimalism in Deductive Databases, *Proceedings of the 5th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 123-132, 1986.

[Bidoit and Legay, 1990] Bidoit, N. and Legay, P., Well !: An Evaluation Procedure for All Logic Programs, *Proceedings of the 3rd International Conference on Database Theory*, Lecture Notes in Computer Sciences 470, Springer-Verlag, 335-348, 1990.

[Bidoit and Froidevaux, 1991a] Bidoit, N. and Froidevaux, C., General Logic Databases and Programs: Default Logic Semantics and Stratification, *Journal of Information and Computation* 91(1), 15-54, 1991.

[Bidoit and Froidevaux, 1991b] Bidoit, N. and Froidevaux, C., Negation by Default and Unstratifiable Logic Programs, *Theoretical Computer Science* 78(1), 85-112, 1991.

[Blair and Subrahmanian, 1989] Blair, H. A. and Subrahmanian, V. S., Paraconsistent Logic Programming, *Theoretical Computer Science* 68, 135-154, 1989.

[Bossu and Siegel, 1985] Bossu, G. and Siegel, P., Saturation, Nonmonotonic Reasoning and the Closed World Assumption, *Artificial Intelligence* 25(1), 13-63, 1985.

[Brass and Dix, 1994] Brass, S. and Dix, J., A Disjunctive Semantics Based on Unfolding and Bottom-up Evaluation, *Proceedings of the 13th IFIP World Computer Congress, GI-Workshop on Disjunctive Logic Programming and Disjunctive Databases*, 1994.

[Bry, 1989] Bry, F., Logic Programming as Constructivism: A Formalization and its Application of Databases, *Proceedings of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 34-50, 1989.

[Cadori and Schaerf, 1993] Cadori, M. and Schaerf, M., A Survey of Complexity Results for Nonmonotonic Logics, *Journal of Logic Programming* 17(2,3& 4), 127-160, 1993.

[Chan, 1993] Chan, E. P. F., A Possible World Semantics for Disjunctive Databases, *IEEE Transactions on Knowledge and Data Engineering* 5(2), 282-292, 1993. Preliminary version in: Research Report CS-89-47, Department of Computer Science, University of Waterloo, 1989.

[Chandra and Harel, 1985] Chandra, A. and Harel, D., Horn Clause Queries and Generalizations, *Journal of Logic Programming* 2(1), 1-15, 1985.

[Chang and Lee, 1973] Chang, C. L. and Lee, R. C. T., *Symbolic Logic and Mathematical Theorem Proving*, Academic Press, New York, 1973.

[Chen, 1993] Chen, J., Minimal Knowledge + Negation as Failure = Only Knowing (sometimes), in [Pereira and Nerode, 1993], 132-150, 1993.

[Chen and Warren, 1993] Chen, W. and Warren, D. S., Query Evaluation under the Well Founded Semantics, *Proceedings of the 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 168-179, 1993.

[Clark, 1978] Clark, K. L., Negation as Failure, in [Gallaire and Minker, 1978], 293-322, 1978.

[Colmerauer et al., 1973] Colmerauer, A., Kanoui, H., Pasero, R., and Ronssel, P., Un Systeme de Communication Homme-Machine en Francais, Research Report, Groupe, Intelligence Artificielle, Universitae de Aix-Marseille II, France, 1973.

[Console et al., 1991] Console, L., Dupre, D.T. and Torasso, P., On the Relationship between Abduction and Deduction, *Journal of Logic and Computation* 1(5), 661-690, 1991.

[da Costa, 1974] da Costa, N. C. A., On the Theory of Inconsistent Formal Systems, *Notre Dame Journal of Formal Logic* 15(4), 497-510, 1974.

[Decker, 1992] Decker, H., Foundations of First-Order Databases, Research Report, Siemens, 1992. Preliminary version in: *Proceedings of the 2nd International Workshop on the Deductive Approach to Information Systems and Databases*, 149-173, Universitat Politecnica de Catalunya, Report de recerca LSI/91/30, 1991.

[Decker and Casamayor, 1993] Decker, H. and Casamayor, J. C., Sustained Models and Sustained Answers in First Order Databases, *Proceedings of the 3rd International Workshop on the Deductive Approach to Information Systems and Databases*, 1993.

[Decker, 1994] Decker, H., Alternative Models and Fixpoints for First Order Databases, Draft Manuscript, 1994.

[Dix, 1992a] Dix, J., Classifying Semantics of Disjunctive Logic Programs, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, MIT Press, 798-812, 1992.

[Dix, 1992b] Dix, J., A Framework of Representing and Characterizing Semantics of Logic Programs, *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann, 591-602, 1992.

[Dung, 1991] Dung, P. M., Negation as Failure for Disjunctive Logic Programming, *Proceedings of the ILPS'91 Post-Conference Workshop on Disjunctive Logic Programs*, 1991.

[Dung and Ruamviboonsuk, 1991] Dung, P. M. and Ruamviboonsuk, P., Well-Founded Reasoning with Classical Negation, in [Nerode et al., 1991], 120-132, 1991.

[Dung, 1992a] Dung, P. M., Acyclic Disjunctive Logic Programs with Abductive Procedure as Proof Procedure, *Proceedings of the International Conference on Fifth Generation Computer Systems*, ICOT, 555-561, 1992.

[Dung, 1992b] Dung, P. M., On the Relations between Stable and Well-founded Semantics of Logic Programs, *Theoretical Computer Science* 105(1), 7-25, 1992.

[Dung, 1993] Dung, P. M., An Abductive Procedure for Disjunctive Logic Programs, Draft Manuscript, 1993.

[Eiter and Gottlob, 1992] Eiter, T. and Gottlob, G., The Complexity of Logic-Based Abduction, Research Report CD-TR 92/35, Technical University Vienna, 1992. To appear in *Journal of the ACM*.

[Eiter and Gottlob, 1993a] Eiter, T. and Gottlob, G., Complexity Aspects of Various Semantics for Disjunctive Databases, *Proceedings of the 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 158-167, 1993.

[Eiter and Gottlob, 1993b] Eiter, T. and Gottlob, G., Complexity Results for Disjunctive Logic Programming and Application to Nonmonotonic Logics, *Proceedings of the International Logic Programming Symposium*, MIT Press, 266-278, 1993.

[Eiter and Gottlob, 1993c] Eiter, T. and Gottlob, G., Propositional Circumscription and Extended Closed World Reasoning are Π_2^P-complete, *Theoretical Computer Science* 114, 231-245, 1993.

[Eiter et al., 1993] Eiter, T., Gottlob, G. and Gurevich, Y., Curb Your Theory ! : A Circumscriptive Approach for Inclusive Interpretation of Disjunctive Information, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, 634-639, 1993.

[Eshghi and Kowalski, 1989] Eshghi, K. and Kowalski, R. A., Abduction Compared with Negation by Failure, *Proceedings of the 6th International Conference on Logic Programming*, MIT Press, 234-254, 1989.

[Eshghi, 1990] Eshghi, K., Computing Stable Models by Using the ATMS, *Proceedings of the 8th National Conference on Artificinl Intelligence*, MIT Press, 272-277, 1990.

[Fages, 1990] Fages, F., Consistency of Clark's Completion and Existence of Stable Models, Research Report, LIENS-90-15, 1990.

[Fages, 1991] Fages, F., A New Fixpoint Semantics for General Logic Programs Compared with the Well-Founded and the Stable Model Semantics, *New Generation Computing* 9(3&4), 425-443, 1991.

[Fernandez and Minker, 1991a] Fernandez, J. A. and Minker, J., Bottom-up Evaluation of Hierarchical Disjunctive Deductive Databases, *Proceedings of the 8th International Conference of Logic Programming*, MIT Press, 660-675, 1991.

[Fernandez and Minker, 1991b] Fernandez, J. A. and Minker, J., Computing Perfect Models of Disjunctive Stratified Databases, *Proceedings of the ILPS'91 Post-Conference Workshop on Disjunctive Logic Programs*, 1991.

[Fernandez and Minker, 1992] Fernandez, J. A. and Minker, J., Disjunctive Deductive Databases, *Proceedings of the International Conference on Logic Programming and Automated Reasoning*, Lecture Notes in Artificial Intelligence 624, Springer-Verlag, 332-356, 1992.

[Fernandez et al., 1993] Fernandez, J. A., Lobo, J., Minker, J. and Subrahmanian, V. S., Disjunctive LP + Integrity Constraints = Stable Model Semantics, *Annals of Mathematics and Artificial Intelligence* 8(3&4), 449-474, 1993.

[Fitting, 1985] Fitting, M., A Kripke-Kleene Semantics for Logic Programming, *Journal of Logic Programming* 2, 295-312, 1985.

[Fitting, 1991] Fitting, M., Bilattices and the Semantics of Logic Programming, *Journal of Logic Programming* 11, 91-116, 1991.

[Fitting, 1993] Fitting, M., The Family of Stable Models, *Journal of Logic Programming* 17(2,3&4), 197-225, 1993.

[Gallaire and Minker, 1978] Gallaire, H. and Minker, J. (eds.), *Logic and Data Bases*, Plenum, New York, 1978.

[Gallaire et al., 1984] Gallaire, H., Minker, J. and Nicolas, J., Logic and Databases: A Deductive Approach, *ACM Computing Surveys* 16(2), 153-185, 1984.

[Garey and Johnson, 1979] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman and Company, NY, 1979.

[Gelfond, 1987] Gelfond, M., On Stratified Autoepistemic Theories, *Proceedings of the 6th National Conference on Artificial Intelligence*, Morgan Kaufmann, 207-211, 1987.

[Gelfond and Lifschitz, 1988] Gelfond, M. and Lifschitz, V., The Stable Model Semantics for Logic Programming, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, MIT Press, 1070-1080, 1988.

[Gelfond et al., 1989] Gelfond, M., Przymusinska, H. and Przymusinski, T., On the Relationship between Circumscription and Negation as Failure, *Artificial Intelligence* 38, 75-94, 1989.

[Gelfond and Lifschitz, 1990] Gelfond, M. and Lifschitz, V., Logic Programs with Classical Negation, *Proceedings of the 7th International Conference on Logic Programming*, MIT Press, 579-597, 1990.

[Gelfond, 1990] Gelfond, M., Epistemic Approach to Formalization of Commonsense Reasoning, Research Report, Computer Science Department, University of Texas at El Paso, 1990.

[Gelfond, 1991] Gelfond, M., Strong Introspection, *Proceedings of the 9th National Conference on Artificial Intelligence*, MIT Press, 386-391, 1991.

[Gelfond and Lifschitz, 1991] Gelfond, M. and Lifschitz, V., Classical Negation in Logic Programs and Disjunctive Databases, *New Generation Computing* 9(3&4), 365-385, 1991.

[Gelfond et al., 1991] Gelfond, M., Lifschitz, V., Przymusinska, H. and Truszczynski, M., Disjunctive Defaults, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann, 230-237, 1991.

[Ginsberg, 1988] Ginsberg, M. L., Multivalued Logics, *Computational Intelligence* 4, 265-316, 1988.

[Gottlob, 1992] Gottlob, G., Complexity Results for Nonmonotonic Logics, *Journal of Logic and Computation* 2(3), 397-425, 1992.

[Grant and Minker, 1986] Grant, J. and Minker, J., Answering Queries in Indefinite Databases and the Null Value Problem, in *Advances in Computing Theory* 3, *The Theory of Databases*, P. Kanellakis (ed.), JAI Press, Greenwich, 247-267, 1986.

[Grant and Subrahmanian, 1992] Grant, J. and Subrahmanian, V. S., Reasoning in Inconsistent Knowledge Bases, draft manuscript, University of Maryland, 1992. To appear in *IEEE Transactions on Knowledge and Data Engineering*.

[Henschen and Park, 1988] Henschen, L. J. and Park, H., Compiling the GCWA in Indefinite Deductive Databases, in [Minker, 1988], 395-438, 1988.

[Inoue, 1991] Inoue, K., Extended Logic Programs with Default Assumptions, *Proceedings of the 8th International Conference on Logic Programming*, MIT Press, 490-504, 1991.

[Inoue et al., 1992] Inoue, K., Koshimura, M. and Hasegawa, R., Embedding Negation as Failure into a Model Generation Theorem Prover, *Proceedings of the 11th International Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence 607, Springer-Verlag, 400-415, 1992.

[Israel, 1983] Israel, D., The Role of Logic in Knowledge Representation, *IEEE Computer* 16(10), 37-42, 1983.

[Johnson, 1990] Johnson, D. S., A Catalog of Complexity Classes, in *Handbook of Theoretical Computer Science*, vol. A, J. van Leeuwen (ed.), Elsevier Science Publishers B. V., North-Holland, 68-161, 1990.

[Kakas and Mancarella, 1990] Kakas, A. C. and Mancarella, P., Generalized Stable Models: A Semantics for Abduction, *Proceedings of the 9th European Conference on Artificial Intelligence*, Pitman, 385-391, 1990.

[Kakas et al., 1992] Kakas, A. C., Kowalski, R. A. and Toni, F., Abductive Logic Programming, *Journal of Logic and Computation* 2(6), 719-770, 1992.

[Kemp and Topor, 1988] Kemp, D. B. and Topor, R. W., Completeness of a Top-Down Query Evaluation Procedure for Stratified Databases, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, MIT Press, 178-194, 1988.

[Kemp et al., 1992] Kemp, D. B., Stuckey, P. J. and Srivastava, D., Query Restricted Bottom-up Evaluation of Normal Logic Programs, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, MIT Press, 288-302, 1992.

[Kifer and Lozinskii, 1992] Kifer, M. and Lozinskii, E. L., A Logic for Reasoning with Inconsistency, *Journal of Automated Reasoning* 9(2), 179-215, 1992.

[Komorowski, 1981] Komorowski, J., A Specification of an Abstract Prolog Machine and its Application to Partial Evaluation, Technical Report LSST 69, Linköping University, 1981.

[Komorowski, 1992] Komorowski, J., An Introduction to Partial Deduction, *Proceedings of the 3rd International Workshop on Meta-programming in Logic*, Lecture Notes in Computer Science 649, Springer-Verlag, 49-69, 1992.

[Konolige, 1988] Konolige, K., On the Relation between Default and Autoepistemic Logic, *Artificial Intelligence* 35, 343-382, 1988. Its Errata, in *Artificial Intelligence* 41, page 115, 1989.

[Kowalski, 1974] Kowalski, R. A., Predicate Logic as a Programming Language, *Information Processing* 74, North-Holland, 569-574, 1974.

[Kowalski, 1979] Kowalski, R. A., Algorithm = Logic + Control, *Communications of the ACM* 22(7), 424-436, 1979.

[Kowalski and Sadri, 1990] Kowalski, R. A. and Sadri, F., Logic Programs with Exception, *Proceedings of the 7th International Conference on Logic Programming*, MIT Press, 598-613, 1990.

[Kowalski, 1991] Kowalski, R. A., Logic Programming in Artificial Intelligence, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, 596-603, 1991.

[Kraus et al., 1990] Kraus, S., Lehmann, D. and Magidor, M., Nonmonotonic Reasoning, Preferential Models and Cumulative Logics, *Artificial Intelligence* 44(1), 167-207, 1990.

[Kunen, 1989] Kunen, K., Signed Data Dependencies in Logic Programs, *Journal of Logic Programming* 7, 231-245, 1989.

[Levesque, 1983] Levesque, H. J., The Logic of Incomplete Knowledge Bases, in *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases, and Programming Languages*, M. Brodie, J. Mylopoulos and J. Schmidt (eds.), Springer-Verlag, NY, 165-186, 1983.

[Lifschitz, 1985] Lifschitz, V., Closed World Database and Circumscription, *Artificial Intelligence* 27, 229-235, 1985.

[Lifschitz, 1988] Lifschitz, V., On the Declarative Semantics of Logic Programs with Negation, in [Minker, 1988], 177-192, 1988.

[Lifschitz, 1989] Lifschitz, V., Between Circumscription and Autoepistemic Logic, *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann, 235-244, 1989.

[Lifschitz and Woo, 1992] Lifschitz, V. and Woo, T. Y. C., Answer Sets in General Nonmonotonic Reasoning (preliminary report), *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann, 603-614, 1992.

[Lifschitz and Schwarz, 1993] Lifschitz, V. and Schwarz, G., Extended Logic Programs as Autoepistemic Theories, in [Pereira and Nerode, 1993], 101-114, 1993.

[Lin and Shoham, 1992] Lin F. and Shoham, Y., A Logic of Knowledge and Justified Assumptions, *Artificial Intelligence* 57, 271-289, 1992.

[Lloyd, 1987] Lloyd, J. W., *Foundations of Logic Programming*, 2nd edition, Springer-Verlag, 1987.

[Lloyd and Shepherdson, 1991] Lloyd, J. W. and Shepherdson, J. C., Partial Evaluation in Logic Programming, *Journal of Logic Programming* 11(3&4), 217-242, 1991.

[Lobo et al., 1989] Lobo, J., Minker, J. and Rajasekar, A., Extending the Semantics of Logic Programs to Disjunctive Logic Programs, *Proceedings of the 6th International Conference on Logic Programming*, MIT Press, 255-268, 1989.

[Lobo and Subrahmanian, 1992] Lobo, J. and Subrahmanian, V. S., Relating Minimal Models and Pre-Requisite-Free Normal Defaults, *Information Processing Letters* 44, 129-133, 1992.

[Lobo et al., 1992] Lobo, J., Minker, J. and Rajasekar, A., *Foundations of Disjunctive Logic Programming*, MIT Press, 1992.

[Lu and Henschen, 1992] Lu, J. J. and Henschen, L. J., The Closed World Assumption in Paraconsistent Deductive Databases, draft manuscript, Northwestern University, 1992.

[Maher, 1993] Maher, M., A Transformation System for Deductive Database Modules with Perfect Model Semantics, *Theoretical Computer Science* 110, 377-403, 1993.

[Manthey and Bry, 1988] Manthey, R. and Bry, F., SATCHMO: A Theorem Prover Implemented in Prolog, *Proceedings of the 9th International Conference on Automated Deduction*, Lecture Notes in Computer Science 310, Springer-Verlag, 415-434, 1988.

[Marek and Truszczynski, 1989a] Marek, W. and Truszczynski, M., Stable Semantics for Logic Programs and Default Theories, *Proceedings of the North American Conference on Logic Programming*, MIT Press, 243-256, 1989.

[Marek and Truszczynski, 1989b] Marek, W. and Truszczynski, M., Relating Autoepistemic and Default Logics, *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann, 276-288, 1989.

[Marek and Truszczynski, 1991a] Marek, W. and Truszczynski, M., Autoepistemic Logic, *Journal of the ACM* 38(3), 588-619, 1991.

[Marek and Truszczynski, 1991b] Marek, W. and Truszczynski, M., Computing Intersection of Autoepistemic Expansions, in [Nerode et al., 1991], 37-50, 1991.

[Marek and Subrahmanian, 1992] Marek, W. and Subrahmanian, V. S., The Relationship between Stable, Supported, Default and Autoepistemic Semantics for General Logic Programs, *Theoretical Computer Science* 103(2), 365-386, 1992.

[McCarthy and Hayes, 1969] McCarthy, J. and Hayes, P., Some Philosophical Problems from the Standpoint of Artificial Intelligence, in *Machine Intelligence* 4, B. Meltzer and D. Michie (eds.), Edinburgh University Press, 463-502, 1969.

[McCarthy, 1980] McCarthy, J., Circumscription – A Form of Nonmonotonic Reasoning, *Artificial Intelligence* 13, 27-39, 1980.

[McCarthy, 1986] McCarthy, J., Applications of Circumscription to Formalizing Commonsense Knowledge, *Artificial Intelligence* 28, 89-118, 1986.

[McDermott and Doyle, 1980] McDermott, D. and Doyle, J., Non-monotonic Logic I, *Artificial Intelligence* 25, 41-72, 1980.

[Minker, 1982] Minker, J., On Indefinite Data Bases and the Closed World Assumption, *Proceedings of the 6th International Conference on Automated Deduction*, Lecture Notes in Computer Science 138, Springer-Verlag, 292-308, 1982.

[Minker, 1988] Minker, J. (ed.), *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann, 1988.

[Minker and Rajasekar, 1990] Minker, J. and Rajasekar, A., A Fixpoint Semantics for Disjunctive Logic Programs, *Journal of Logic Programming* 9, 45-74, 1990.

[Moore, 1985] Moore, R. C., Semantical Considerations on Nonmonotonic Logic, *Artificial Intelligence* 25, 75-94, 1985.

[Neches et al., 1991] Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T. and Swartout, W. R., Enabling Technology for Knowledge Sharing, *AI Magazine* 12(3), 36-56, 1991.

[Nerode et al., 1991] Nerode, A., Marek, W., and Subrahmanian, V. S. (eds.), *Logic Programming and Nonmonotonic Reasoning, Proceedings of the 1st International Workshop*, MIT Press, 1991.

[Peirce, 1932] Peirce, C. S., *Elements of Logic*, in *Collected Papers of Charles Sanders Peirce* II, Hartshorn et al. (eds.), Harvard University Press, Cambridge, 1932.

[Pereira et al., 1991] Pereira, L. M., Alferes, J. J. and Aparicio, N., Contradiction Removal within Well-Founded Semantics, in [Nerode et al., 1991], 105-119, 1991.

[Pereira and Alferes, 1992] Pereira, L. M. and Alferes, J. J., Well-Founded Semantics for Logic Programs with Explicit Negation, *Proceedings of the 10th European Conference on Artificial Intelligence*, Wiley, 102-106, 1992.

[Pereira and Nerode, 1993] Pereira, L. M. and Nerode, A. (eds.), *Logic Programming and Nonmonotonic Reasoning, Proceedings of the 2nd International Workshop*, MIT Press, 1993.

[Pimentel and Rodi, 1991] Pimentel, S. G. and Rodi, W. L., Belief Revision and Paraconsistency in a Logic Programming Framework, in [Nerode et al., 1991], 228-242, 1991.

[Poole, 1989] Poole, D., What the Lottery Paradox Tells us about Default Reasoning, *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann, 333-340, 1989.

[Przymusinski, 1988a] Przymusinski, T. C., On the Declarative Semantics of Deductive Databases and Logic Programs, in [Minker, 1988], 193-216, 1988.

[Przymusinski, 1988b] Przymusinski, T. C., On the Relationship between Nonmonotonic Reasoning and Logic Programming, *Proceedings of the 7th National Conference on Artificial Intelligence*, Morgan Kaufmann, 444-448, 1988.

[Przymusinski, 1989a] Przymusinski, T. C., Every Logic Program has a Natural Stratification and an Iterated Least Fixed Point Model, *Proceedings of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems*, 11-21, 1989.

[Przymusinski, 1989b] Przymusinski, T. C., An Algorithm to Compute Circumscription, *Artificial Intelligence* 38, 49-73, 1989.

[Przymusinski, 1989c] Przymusinski, T. C., On the Declarative and Procedural Semantics of Logic Programs, *Journal of Automated Reasoning* 5, 167-205, 1989.

[Przymusinski, 1990a] Przymusinski, T. C., Extended Stable Semantics for Normal and Disjunctive Programs, *Proceedings of the 7th International Conference on Logic Programming*, MIT Press, 459-477, 1990.

[Przymusinski, 1990b] Przymusinski, T. C., Stationary Semantics for Disjunctive Logic Programs and Deductive Databases, *Proceedings of the North American Conference on Logic Programming*, MIT Press, 40-62, 1990.

[Przymusinski, 1990c] Przymusinski, T. C., The Well-founded Semantics Coincides with the Three-Valued Stable Semantics, *Fundamenta Informaticae* 13, 445-464, 1990.

[Przymusinski, 1991a] Przymusinski, T. C., Stable Semantics for Disjunctive Programs, *New Generation Computing* 9(3&4), 401-424, 1991.

[Przymusinski, 1991b] Przymusinski, T. C., Semantics of Disjunctive Logic Programs and Deductive Databases, *Proceedings of the 2nd International Conference on Deductive and Object-Oriented Databases*, Lecture Notes in Computer Science 566, Springer-Verlag, 85-107, 1991.

[Przymusinski, 1991c] Przymusinski, T. C., Three-valued Nonmonotonic Formalisms and Semantics of Logic Programs, *Artificial Intelligence* 49, 309-343, 1991.

[Rajasekar et al., 1989] Rajasekar, A., Lobo, J. and Minker, J., Weak Generalized Closed World Assumption, *Journal of Automated Reasoning* 5, 293-307, 1989.

[Rajasekar and Minker, 1989] Rajasekar, A. and Minker, J., A Stratification Semantics for General Disjunctive Programs, *Proceedings of the North American Conference on Logic Programming*, MIT Press, 573-586, 1989.

[Reed et al., 1991] Reed, D.W., Loveland, D.W. and Smith, B.T., An Alternative Characterization of Disjunctive Logic Programs, *Proceedings of the International Logic Programming Symposium*, MIT Press, 54-68, 1991.

[Reiter, 1978] Reiter, R., On Closed World Databases, in [Gallaire and Minker, 1978], 55-76, 1978.

[Reiter, 1980] Reiter, R., A Logic for Default Reasoning, *Artificial Intelligence* 13, 81-132, 1980.

[Reiter, 1982] Reiter, R., Circumscription implies Predicate Completion (Sometimes), *Proceedings of the 2nd National Conference on Artificial Intelligence*, William Kaufmann, 418-420, 1982.

[Robinson, 1965a] Robinson, J., A Machine-Oriented Logic Based on the Resolution Principle, *Journal of the ACM* 12, 23-41, 1965.

[Robinson, 1965b] Robinson, J., Automatic Deduction with Hyper-Resolution, *Journal of Computer Mathematics* 1, 227-234, 1965.

[Ross and Topor, 1988] Ross, K. A. and Topor, R. W., Inferring Negative Information from Disjunctive Databases, *Journal of Automated Reasoning* 4, 397-424, 1988.

[Ross, 1989a] Ross, K., A Procedural Semantics for Well Founded Negation in Logic Programs, *Proceedings of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 22-33, 1989.

[Ross, 1989b] Ross, K., The Well Founded Semantics for Disjunctive Logic Programs, *Proceedings of the 1st International Conference on Deductive and Object-Oriented Databases*, North-Holland, 385-402, 1989.

[Sacca and Zaniolo, 1990] Sacca, D. and Zaniolo, C., Stable Models and Nondeterminism in Logic Programs with Negation, *Proceedings of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 205-229, 1990.

[Sadri and Kowalski, 1988] Sadri, F. and Kowalski, R., A Theorem-Proving Approach to Database Integrity, in [Minker, 1988], 313-362, 1988.

[Sato, 1990] Sato, T., Completed Logic Programs and their Consistency, *Journal of Logic Programming* 9(1), 33-44, 1990.

[Satoh and Iwayama, 1991] Satoh, K. and Iwayama, K., Computing Abduction by using the TMS, *Proceedings of the 8th International Conference on Logic Programming*, MIT Press, 505-518, 1991.

[Schlipf, 1992a] Schlipf, J. S., Formalizing a Logic for Logic Programming, *Annals of Mathematics and Artificial Intelligence* 5, 279-302, 1992.

[Schlipf, 1992b] Schlipf, J. S., A Survey of Complexity and Undecidability Results in Logic Programming, *Proceedings of the JICSLP'92 Post-Conference Workshop on Structural Complexity and Recursion-Theoretic Methods in Logic Programming*, 143-164, 1992.

[Seki and Itoh, 1988] Seki, H. and Itoh, H., A Query Evaluation Method for Stratified Programs under the Extended CWA, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, MIT Press, 195-211, 1988.

[Seki, 1990] Seki, H., A Comparative Study of the Well-Founded and the Stable Model Semantics: Transformation's Viewpoint, *Proceedings of the NACLP'90 Post-Conference Workshop on Logic Programming and Nonmonotonic Reasoning*, 115-123, 1990.

[Seki, 1991] Seki, H., Unfold/Fold Transformation of Stratified Programs, *Theoretical Computer Science* 86, 107-139, 1991.

[Seki, 1993] Seki, H., Unfold/Fold Transformation of General Logic Programs for the Well-Founded Semantics, *Journal of Logic Programming* 16(1), 5-23, 1993.

[Selman and Levesque, 1990] Selman, B. and Levesque, H. J., Abductive and Default Reasoning: A Computational Core, *Proceedings of the 8th National Conference on Artificial Intelligence*, MIT Press, 343-348, 1990.

[Sestoft and Zamulin, 1988] Sestoft, P. and Zamulin, A. V., Annotated Bibliography on Partial Evaluation and Mixed Computation, *New Generation Computing* 6(2&3), 309-354, 1988.

[Shepherdson, 1984] Shepherdson, J., Negation as Failure: A Comparison of Clark's Completed Data Base and Reiter's Closed World Assumption, *Journal of Logic Programming* 1, 51-79, 1984.

[Shepherdson, 1988] Shepherdson, J., Negation in Logic Programming, in [Minker, 1988], 19-88, 1988.

[Subrahmanian, 1992] Subrahmanian, V. S., Paraconsistent Disjunctive Deductive Databases, *Theoretical Computer Science* 93, 115-141, 1992.

[Subrahmanian et al., 1993] Subrahmanian, V. S., Nau, D. and Vago, C., WFS + Branch and Bound = Stable Models, Research Report, University of Maryland, 1993.

[Tamaki and Sato, 1984] Tamaki, H. and Sato, T., Unfold/Fold Transformation of Logic Programs, *Proceedings of the 2nd International Conference on Logic Programming*, 127-138, 1984.

[Tärnlund, 1977] Tärnlund, S. -A., Horn Clause Computability, *BIT* 17(2), 215-226, 1977.

[Teusink, 1993a] Teusink, F., A Characterization of Stable Models using a Non-monotonic Operator, in [Pereira and Nerode, 1993], 206-222, 1993.

[Teusink, 1993b] Teusink, F., A Proof Procedure for Extended Logic Programs, *Proceedings of the International Logic Programming Symposium*, MIT Press, 235-249, 1993.

[Ullman, 1982] Ullman, J. D., *Principles of Database Systems*, 2nd edition, Computer Science Press, 1982.

[van Emden and Kowalski, 1976] van Emden, M. H. and Kowalski, R. A., The Semantics of Predicate Logic as a Programming Language, *Journal of the ACM* 23(4), 733-742, 1976.

[van Gelder, 1988] van Gelder, A., Negation as Failure Using Tight Derivations for General Logic Programs, in [Minker, 1988], 149-176, 1988.

[van Gelder, 1989] van Gelder, A., The Alternating Fixpoint of Logic Programs with Negation, *Proceedings of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1-10, 1989.

[van Gelder et al., 1991] van Gelder, A., Ross, K. and Schlipf, J. S., The Well-Founded Semantics for General Logic Programs, *Journal of the ACM* 38(3), 620-650, 1991.

[Wagner, 1991a] Wagner, G., A Database Needs Two Kinds of Negation, *Proceedings of the 3rd Symposium on Mathematical Fundamentals of Database and Knowledge Base Systems*, Lecture Notes in Computer Science 495, Springer-Verlag, 357-371, 1991.

[Wagner, 1991b] Wagner, G., Ex contradictione nihil sequitur, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, 538-543, 1991.

[Wagner, 1993] Wagner, G., Reasoning with Inconsistency in Extended Deductive Databases, in [Pereira and Nerode, 1993], 300-315, 1993.

[Yahya and Henschen, 1985] Yahya, A. and Henschen, L. J., Deduction in Non-Horn Databases, *Journal of Automated Reasoning* 1(2), 141-160, 1985.

[Yuan and You, 1993] Yuan, L. Y. and You, J-H., Autoepistemic Circumscription and Logic Programming, *Journal of Automated Reasoning* 10(2), 143-160, 1993.